AMOBA - ARINC 653 SIMULATOR FOR MODULAR BASED SPACE APPLICATIONS

Edgar Pascoal[1], José Rufino*[2], Tobias Schoofs[3], and James Windsor[4]

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

ABSTRACT

The ARINC 653 standard has taken a leading role within the aeronautical industry in the development of safety-critical systems based upon the Integrated Modular Avionics (IMA) concept. The related cost savings in reduced integration, verification and validation effort has raised interest in the European space industry for developing a spacecraft IMA approach and for the definition of an ARINC 653-for-space software framework. As part of this process, it is necessary to establish an effective way to test and develop space applications without having access to the final IMA target platform. This paper describes the design and the architecture of a multi-platform and modular ARINC 653 simulator that emulates an execution environment for ARINC 653 space applications.

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

1. INTRODUCTION

The ARINC 653 [1, 2] is a standard that specifies a programming interface for a RTOS (Real-Time Operating System), and, in addition, establishes a particular method for partitioning resources over time and memory. At the moment this standard has been established as an important foundation for the development of safety-critical systems in the avionics industry. This fact allied to several similarities regarding the development of space on-board software has aroused attention of ESA (European Space Agency) to built a new concept based on the IMA (Integrated Modular Avionics) philosophy adapted for space domain [3, 4, 5, 6].

The ARINC 653 standard [1, 2] based approach for space purposes shall bring new ways and concepts that have to be explored and improved. Therefore, it could be useful to have a tool to help on this discovering process, as well as provides a test and develop environment for on-board applications without needing access to the final target platform.

The AMOBA concept will take advantage of the ARINC 653 [1, 2] and Portable Operating System Interface (POSIX) [7] standards to obtain a multi-platform simulation environment for the development of safety critical software for space domain. Thus, the simulator environment will allow the execution over several platforms of application programs developed in the context of IMA [3] by offering an emulation of the ARINC 653 IMA Application Programming Interface (API), also known as APEX (APplication EXecutive).

2. SPACE REQUIREMENTS

In order to create the new ARINC 653-for-space concept it is needed to identify space requirements; the differences between the avionics and space markets must be carefully studied.

Avionics concepts will probably not be applied one-to-one to the space context; they will undergo certain modifications, improvements or pure alterations. Discussions and research activities in the aeronautical context are also addressing potential improvements in the IMA design. Some of the proposed improvements appear to point in a very similar direction as the discussion related to space. The modifications IMA has to undergo to fit into space needs are probably also necessary for future avionics requirements [8, 2, 9].

This section discusses some of the differences and the resulting issues. AMOBA is not able to solve all these issues. But it is a good occasion to study them and the AMOBA simulator is a powerful mean to develop solutions for space requirements and to prove their feasibility.

The main differences between space and aeronautic contexts follow from three characteristics of space missions:

- their duration – be it a spacecraft on a journey to the outer planets, be it a satellite orbiting earth – which

*The contribution to this work was partially supported by EU and FCT, through Project POSC/EIA/56041/2004 (DARIO) and through the FCT Multiannual Funding Program.

makes it impossible to update or maintain the computer systems physically in a ground based hangar;

- the lack of personnel on board to intervene in critical situations or to change mission programs;
- the distance from ground which makes it difficult when not impossible to intervene in time from ground control, not to mention the impossibility of an emergency landing [8].

These characteristics imply **autonomy**, **flexibility** and the ability of **remote control** and security as core features of systems in space.

Autonomy – means that the system is able to perform short time decisions on its own. It must be able to control itself and the proceeding of the mission, to take decisions on the base of this analysis and to perform the actions which follow. The system must be enabled to draw conclusions of hardware or material failures, to substitute a running component with a redundant one, e.g. as repair after physical damage [10].

Flexibility – means that the system is reconfigurable during runtime. A more dynamic configuration concept is needed than that proposed by the ARINC 653 standard. Reconfiguration may imply the fall-back to a reduced operational mode to guarantee the realisation of the main goals of the mission dropping additional tasks in case of emergency [10, 2].

Remote control and security – means that the system can be controlled remotely. The lack of pilots and system administrators in unmanned missions requires the possibility to control the system remotely. Remote control demands communication channels from ground control not only to the control system but also to the payload applications. For this a robust security concept is necessary to protect the space mission from non-authorized players [11, 12].

3. ARINC 653 ARCHITECTURE

The general architecture of a standard ARINC 653 system is illustrated in Figure 1. It comprises the application software layer, with each application executing in a dedicated partition, and a given set of system partitions. The system partitions are optional components and are intended to manage the interactions with specific hardware devices. Appropriate support from the core software layer (e.g. hardware interfacing and device drivers) is required.

Application partitions consist in general of one or more processes that exclusively use the services provided by a logical Application Executive (APEX) interface, meaning calls can only be made to the set of primitives defined in the ARINC 653 standard specification [1]. However, a system partition may use also a set of specific functions

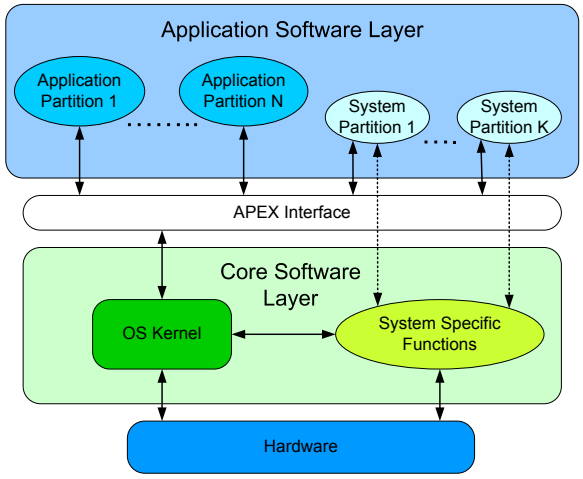


Figure 1. Overview of the Standard ARINC 653 System Architecture

provided by the core software layer and may therefore bypass the standard APEX interface, as illustrated by the dashed arrows in Figure 1.

In any case, the execution environment provided by the Operating System (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. The application software layer is obliged to strict robust space and time partitioning. Containment of possible faults inside the domain of each partition must be ensured in the core software layer [1].

Thus, each partition makes use of a logical execution environment. Given most of the process-level services that need to be supplied by the OS kernel module are already provided by common off-the-shelf real-time operating system (RTOS) kernels, a natural approach to the definition and design of ARINC 653 systems may use the functionality provided by a given RTOS kernel (e.g. RTEMS [13]), completed with the specific functions needed for ARINC 653 system operation, namely partition management mechanisms [5, 6].

The execution environment of each partition includes the corresponding application program (code and data), its configuration attributes and execution context (e.g. stack).

4. AMOBA: ARINC 653 SIMULATOR

The AMOBA is a multi-platform ARINC 653 simulator aimed to be used for the development and verification of space applications. This section defines the AMOBA design goals, describes the simulator's architecture and discusses a relevant set of typical use-case scenarios.

4.1. AMOBA Design Concept

The main purpose of AMOBA is to provide to users an execution environment with the capability to execute and verify ARINC 653 applications. 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 RTOS.

Another fundamental design attribute of AMOBA concerns the portability of the simulator. A multi-platform approach shall provide availability of simulator on every POSIX-compliant operating system. Additionally, modularity is an essential design requirement for the application of the simulator to various use-cases (see §4.3).

In addition to allowing verification of ARINC 653 applications over any POSIX-compliant operating systems, AMOBA shall also provide extra capabilities to the user like logging time measurements, and overall system monitoring, such as detecting and reporting failures. In this sense, AMOBA emulates and extends the scope of the functions assigned to the Health Monitor component foreseen on the ARINC 653 standard [1].

4.2. AMOBA Architecture

The AMOBA Simulator is intended to be multi-platform, so it must avoid dependency between APEX and the OS kernel (cf. Figure 1), which means that it shall not access directly the given OS kernel interface unless it is strictly necessary. To avoid and control that possible dependency AMOBA foresees a middle abstraction layer, called "AMOBA-Core", which provides support to a "virtual" ARINC 653 execution environment [14].

The AMOBA architecture follows a layered design approach as shown in Figure 2, where the simulator has the following main modules:

AMOBA-APEX Interface – is the APEX interface implementation within the AMOBA simulator and its main objective 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 is completely based on the AMOBA-Core layer – the APEX can be ported to another system without any changes. All platform-specific functionality is hidden in the Core-layer. The APEX-Interface aims to provide services for:

- partition management;
- process management;
- time management;
- intra-partition communication;
- inter-partition communication.

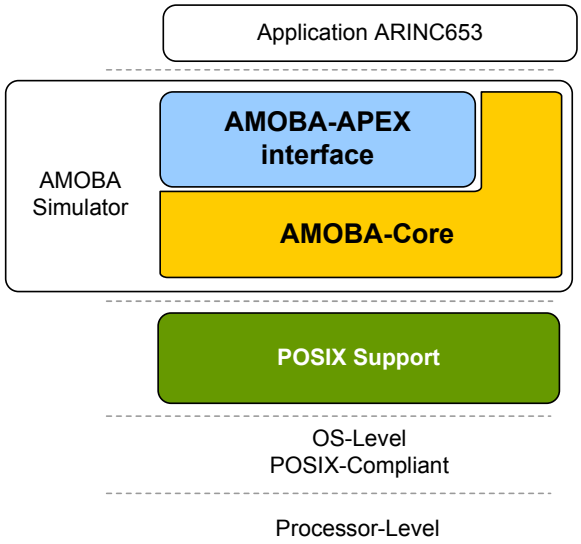


Figure 2. AMOBA: ARINC 653 Simulator Architecture

AMOBA-Core – this is the central part of the AMOBA simulator and its main objective is to provide all OS necessary services to upper layers making use of a POSIX interface (e.g. RTEMS POSIX API [15]). The following components have been identified to be included in the AMOBA-Core layer:

- configuration management;
- time-measurement and time/function monitoring;
- ARINC 653 functional emulation;
- process scheduling;
- inter-partition communication means.

The configuration management, time-measurement and time/function monitoring components are mandatory in the AMOBA architecture and must always be present. The ARINC 653 functional components are intended to provide the services needed for APEX implementation. They hide the platform specifics from the APEX level – such that the APEX can be ported even to non-POSIX systems without any changes. The necessary changes will be limited to the AMOBA-Core layer.

This design also strengthens flexibility with regard to where the ARINC 653 functional emulation is located. If the underlying RTOS kernel already provides these functions the AMOBA-Core only has to contribute with the corresponding monitoring functions. If not then it has to provide the functionality itself.

AMOBA-Core: Process Scheduler – the objective of this module is to implement a process scheduler for the AMOBA-Core that is safe and simple in terms of code, exhibiting low processing overheads and providing strict compliance with the ARINC 653 standard [1, 2].

The chosen solution is an encapsulated component that uses features of the underlying OS, but following known scheduler design approaches [16].

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

- Dormant - process ineligible for scheduling;
- Ready - process is able to be executed;
- Running - process is currently executing;
- Waiting - process is not able to execute.

The APEX processes are maintained in lists according to these states. The states are now mapped to priorities in the POSIX thread library such that the running process has a higher priority than the processes in the other lists. To avoid that a non-running process starts executing an idle process is inserted with a priority less than the running priority and greater than the non-running priority.

AMOBA-Core: Time Manager Module – the implementation of time management addressed ARINC 653 standard definition [1]: *"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 AMOBA-Core layer, in the sense it shall provide all the fundamental time related support for its own layer and support "time-measurement and time-monitoring" services to the above layers. The main services provided by the Time Manager are:

- a function to retrieve the current system time;
- a wait, alarm and time-out mechanism;
- and a deadline observer.

The time manager provides interfaces to register and cancel events, like alarms, time-outs or deadlines. The time manager always waits for the next event. If a newly registered event is earlier than the earliest in the queue, the time manager is interrupted to point to the new one.

A granularity can be set during initialisation that defines the smallest possible wait-interval. This way, the simulator can be optimised for different platforms, operating systems and applications.

AMOBA-Core: Partition Manager Module – the purpose of the partition manager is to provide all the partition related operations in order to supply all the needed functionality to the AMOBA-Core. Because of the services characteristics that this component has to provide, it is separated into three components with the purpose of improving component's modularity and independency. Independency among each of those components shall be preserved:

- Partition Scheduler – its role is to provide a partition execution environment. Meaning that, it shall control the execution of each partition regarding the schedule plan previously defined on configuration time by the system integrator.
- Partition Services - This component has two purposes:
  - ensure that all partition services required are available to AMOBA-APEX so that it could implement the APEX interface according to the ARINC 653 standard [1];
  - guarantee that internal partition data is available to the Partition Scheduler so that this manager can create a scheduling execution environment.
- Inter-partition Communication - this component aims to provide all the necessary inter-partition communication services and their infrastructure to AMOBA-APEX interface, in conformity with the defined in the ARINC 653 standard [1].

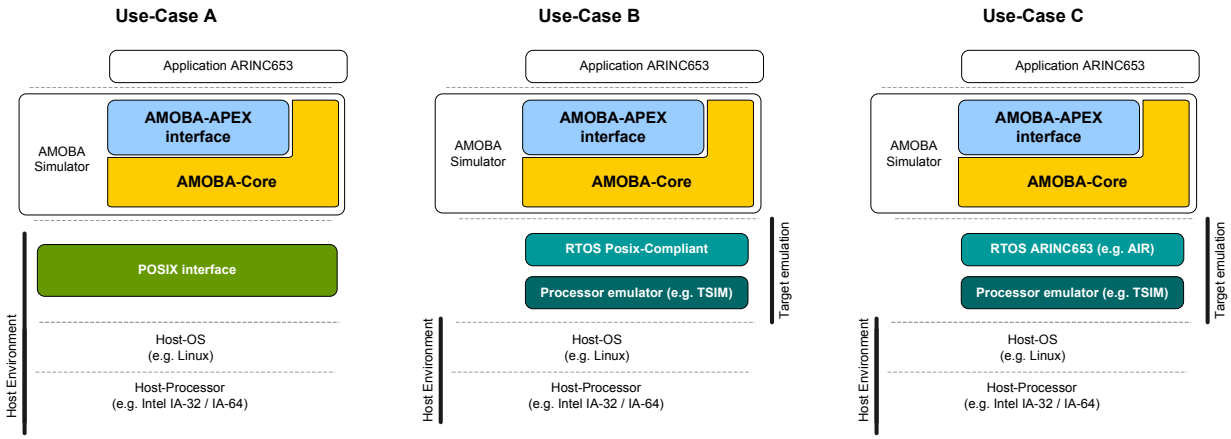
The partition scheduler, again, is based on the POSIX thread library. The active process in the currently executing partition is mapped to the priority defined by the process scheduler, whereas the active process of inactive partitions is mapped to the non-running priority. The context switch between two partitions is, thus, minimal: the non-running priority is set to the running process of the old partition; in the new partition the priority of the running process is restored.

4.3. AMOBA Use-Cases

The relation between the AMOBA-Core components and the OS must be as much generic as possible. As such, AMOBA-Core components that access OS services shall be implemented as POSIX-based primitives in order to support a high number of compatible operating systems. Most of the currently available general-purpose operating systems (e.g. Linux) and RTOS kernels (e.g. RTEMS [13, 15], eCos [17], VxWorks [18], etc...) are POSIX compliant. The improved OS services, minimizing the direct dependency between the AMOBA simulator and the OS, allow more portability to AMOBA simulator.

Different execution environments may be defined for the AMOBA Simulator, as illustrated in Figure 3. An ARINC 653 functional emulation uses the full AMOBA-Core to model the end target system, being executed directly on a POSIX-compliant host environment [22, 23], as shown in the Use-Case A of Figure 3.

Emulation environments (Use-Cases B and C of Figure 3) assume the utilization of target processor virtualization. For on-board space systems the utilization of SPARC LEON target platforms is assumed, emulated for instance through TSIM2 [20] or SIMULUS/LeonVM [21].



ARINC 653 Functional Emulation	ARINC 653 Target Emulation	
AMOBA-Core	AMOBA-Core	
full ARINC 653 functional emulation	some ARINC 653 functional emulation	no ARINC 653 functional emulation
Development Environment	Development Environment	
host compiler toolchain POSIX support library (e.g. [19])	target processor cross-compiler toolchain target processor emulator (e.g. [20, 21])	

Figure 3. AMOBA Simulator and typical use-case scenarios

At operating system level, a RTOS kernel can be used to partially provide ARINC 653 run-time environment functions (e.g. process scheduling and management) [1, 13, 15], as illustrated in the Use-Case B of Figure 3. In addition, the AMOBA Simulator may be used with off-the-shelf RTOS implementations of the ARINC 653 specification, such as the technology being developed within the scope of the AIR Project [6, 24]. This is shown in the Use-Case C of Figure 3. The added-value of this approach is the monitoring of the overall system in an emulated, yet very realistic environment.

In target emulated environments, time-related analysis (e.g. verification of timeliness properties and overall system validation in the time domain) may be assisted by additional methods and tools. For example, the method described in [25] allows to analyse the feasibility of scheduling a given process set, integrating both periodic and sporadic processes, by a priority-based preemptive algorithm and to determine the corresponding worst-case process response times. The extension of such method to ARINC 653 systems is further analised in [26].

The functionality provided by the AMOBA-Core, in the diagram of Figure 3, decreases from Use-Case A to C whereas the functionality of the emulation environment increases in this direction.

5. CONCLUDING REMARKS

Since ARINC 653 has proven to be a good way to develop on-board software in the aeronautical avionics market, the space industry has shown interested in extending the ARINC 653 to the space domain but not limited to it. To enable the adoption of this new architecture it is necessary to have in place for the space industry a toolset to assist in the development and verification of ARINC 653 based systems.

The AMOBA simulator is the first tool development for the ARINC 653 concept in the space domain. AMOBA provides the end-user with the possibility to verify space on-board software applications by using a modular simulator that will be deployable on any POSIX-compliant operating system and offers extra capabilities to verify the application's behaviour.

In the aeronautical context, AMOBA is already used as development tool in the European Commission 6th Framework Project DIANA (Distributed equipment Independent environment for Advanced avionic Applications) [9] as well as in the development of Skysoft's AVT (ARINC 653 Verification Tool) [27].

In space, it will be used to support further research and development in the application of IMA and ARINC 653 application to space.

This paper has described the main issues regarding the AMOBA concept and the simulator's architecture. The AMOBA architecture is POSIX-based and it is aimed to be portable to any operating system supporting a POSIX interface, opening room for several use-case scenarios, ranging from a simple functional simulation to a complete target emulation of ARINC 653 execution environments.

REFERENCES

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

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

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

[4] 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.

[5] 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.

[6] 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.

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

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

[9] DIANA whitepaper. Skysoft S.A., Lisboa, Portugal, March 2008. `http://www.dianaproject.com`.

[10] E. J. Ruggiero. *Modeling and Control of SPIDER Satellite Components*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA., July 2005.

[11] J. N. Pelton. Satellite security and performance in an era of dual use. *Online Journal of Space Communication*, 6, 2004. `http://satjournal.tcom.ohiou.edu`.

[12] J. Alves-Foss, W. S. Harrison, P. Oman, and C. Taylor. The MILS architecture for high-assurance embedded systems. *International Journal for Embedded Systems*, 2(3-4):239–247, August 2006.

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

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

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

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

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

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

[19] R. Engelschall. The GNU portable threads, 2006.

[20] Gaisler Research, Goteborg, Sweden. *TSIM2 Simulator Users Manual (Version 2.0.9)*, April 2008.

[21] P. Marques, J. Feiteirinha, L. Pureza, N. Lindman, and M. Pecchioli. LeonVM: Using dynamic translation for developing high-speed space processor emulators. In *Proceedings of the 9th Int. Workshop on Simulation for European Space Programmes (SESP'2006)*, Noordwijk, The Netherlands, November 2006. ESA/ESTEC.

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

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

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

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

[26] M. Coutinho. Integração modular de dispositivos de entrada/saída em plataformas de controlo distribuído. Master's thesis, Instituto Superior Técnico, Technical University of Lisbon, Lisboa, Portugal, December 2007. (in portuguese).

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

[28] Timekeeping in VMware virtual machines. White Paper, 2005.