Adaptability and Survivability in Spaceborne Time- and Space-Partitioned Systems

Joaquim Rosa, João Craveiro, and José Rufino
University of Lisbon, Faculty of Sciences, LaSIGE
FCUL, Ed. C6, Piso 3, Campo Grande, 1749-016 Lisbon, Portugal
{jrosa, jcraveiro}@lasige.di.fc.ul.pt, ruf@di.fc.ul.pt

***Abstract*—Future space missions call for innovative computing system architectures, meeting strict requisites of size, weight, power consumption (SWaP), cost, safety and timeliness. To answer the aerospace industry interests, especially the European Space Agency (ESA), the AIR (ARINC 653 in Space Real-Time Operating System) architecture has been defined, following the advanced notion of time and space partitioning (TSP). In order to achieve adaptability and survivability in the face of environmental changes or new mission requirements, it is of paramount importance that onboard computing systems are reconfigurable. In this paper we present recent and ongoing developments on AIR Technology to achieve adaptability and survivability of AIR-based systems, and a methodology for onboard software update in order to add new features to the mission plan.**

I. Introduction

To face future challenges, spaceborne onboard computing system architectures should be designed to improve the spacecraft survivability and prolong the mission activity, while obeying to strict dependability and real-time requisites. Partitioned architectures implementing the logical separation of applications in criticality domains, named partitions, allow several partitions to share the same computational infrastructure and fulfil the size, weight and power consumption (SWaP) requisites, thus decreasing the overall cost of the mission [1], [2]. The notion of time and space partitioning (TSP) means ensuring that the execution of an application in one partition does not affect other partitions' timing requirements and that different addressing spaces are assigned to different partitions. The AIR Technology answers the interest from the space industry in applying the TSP concepts to the space domain, defining a partitioned architecture for the development of aerospace systems and applications [3].

It has been shown that reconfigurability and software upgradeability can be crucial to the survival of a mission. For example, many spacecraft exceeding their expected lifetime have been recovered after applying healing procedures upon the occurrence of incidents which could otherwise result in mission degradation or total spacecraft loss [4]. The onboard software update methodology addressed in this paper takes advantage of the composability properties of the AIR architecture aiming the definition of a software development and update process for adaptive TSP systems.

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.

This paper is organized as follows. Section II describes the AIR architecture. Section III explains how to take advantage of AIR schedulability and composability. Section IV addresses adaptability and survivability properties and mechanisms. Section V highlights the current developments to support onboard software update. Finally, Section VI issues concluding remarks and presents future research directions.

II. AIR Technology

The AIR Technology is currently evolving to the definition of an industrial product towards the improvement and completeness of the architecture design and engineering process.

A. System architecture

The AIR architecture, illustrated in Fig. 1, relies on the *AIR Partition Management Kernel* (PMK) to enforce robust TSP. A (real-time or generic) operating system, herein referred as *Partition Operating System* (POS), is provided per partition. Each POS is wrapped by the *AIR POS Adaptation Layer* (PAL) [3] hiding its particularities from other AIR components.

An AIR-based system provides a way to achieve the containment of faults to the domain where they occur using the architectural principle of robust TSP. Temporal partitioning ensures that the real-time requisites of the different functions executing in each partition are guaranteed. Spatial partitioning relies on having dedicated memory and input/output (I/O) addressing spaces for applications executing on different partitions.

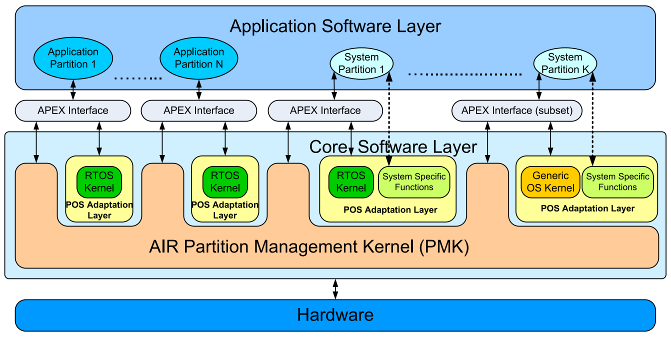


Fig. 1. AIR system architecture and integration of partition operating systems

B. *Portable Application Executive (APEX) Interface*

The *Application Executive* (APEX) *interface* component provides a standard programming interface with a service definition derived from the ARINC 653 specification [5]. The set of available services concerns partition and process management, time management, intra and interpartition communication and health monitoring. The AIR architecture implements the advanced notion of *Portable* APEX, ensuring portability between the different POSs, exploiting the availability of AIR PAL wrapped functions.

C. *AIR Health Monitor*

The AIR architecture also incorporates a *Health Monitor* (HM) component which is responsible for handling and containing errors to their domains of occurrence. This component plays an important role in achieving system adaptability since it prevents errors detected in one process/partition from propagating to the remaining partitions. The action to be performed in the event of an error is defined by the application programmer through an appropriate error handler. This error handler is an application process which should include a systemwide reconfigurability logic, which helps achieve system adaptability and may comprise the redefinition of control parameters or the issue of a different schedule request.

D. *Interpartition communication*

The organization of spacecraft software components in different partitions requires interpartition communication facilities, since a function hosted in a partition may need to exchange information with other partitions. Interpartition communication consists of the authorized transfer of information between partitions without violating spatial separation constraints [3].

III. Schedulability and Composability

The AIR architecture uses a two-level scheduling scheme, where partitions are scheduled under a predetermined sequence of time windows, cyclically repeated over a *major time frame* (MTF). In each partition, the respective processes are scheduled according to the native operating system's process scheduler (Fig. 2).

The original ARINC 653 [5] notion of a single fixed partition scheduling table, defined offline, is limited in terms of timeliness control and fault tolerance. To address this limitation, the AIR Technology design incorporates the notion of *mode-based partition schedules* (Fig. 2), allowing the switch between different partition scheduling tables (PSTs) according to different mission phases or operating modes during the execution time, and regarding the accommodation of component failures [3], [6]. A schedule switch can be requested by a specifically authorized and certified partition through the invocation of an APEX primitive [7]. This can result from a command issued from ground control or from the reaction to environmental conditions. The *AIR Partition Scheduler* is responsible for guaranteeing to make a schedule switch effective at the end of the respective MTF.

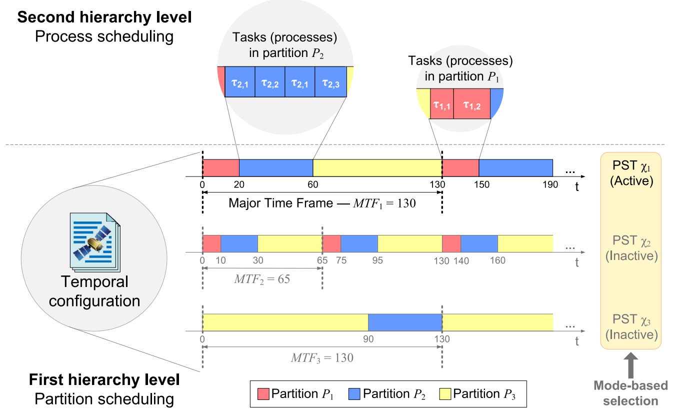


Fig. 2. Two-level mode-based partition scheduling

The modularity of the AIR architecture design and of its build and integration process further enables the *composability* of AIR-based systems [8]. Composability means properties established for individual components hold also after the components are assembled together into the system. This allows the independent verification and validation of software components to different software developers during the build and integration process, facilitating the overall system certification. From the point of view of one partition's provider, this further signifies that development and validation do not depend on knowledge of the other partitions. The system integrator is responsible for guaranteeing a correct partition scheduling, making use of schedulability analysis tools [8], [9].

IV. Adaptability and Survivability

Adaptation to changing or unexpected conditions is of great importance for a mission's survival since it has been proven to prolong the lifetime of unmanned space vehicles by years [4].

A. *Reconfigurability*

The AIR architecture provides support for adaptability and survivability through several mechanisms, such as the mode-based schedules, allowing the definition and switch between multiple PSTs; process deadline violation monitoring, to detect deadline violations in relation to the partitions' timing requirements; AIR Health Monitor, processing partition/process errors, and; low-level event overload control through AIR PMK adaptation mechanisms to control the timeliness of asynchronous events [10], [3]. The support for onboard software update in the AIR architecture [11], described in Section V, plays an important role in achieving the system adaptability and survivability, since it enables the update of system functions and PSTs.

B. *Achieving Adaptability*

Adaptation reflects the capability of maintaining or improving system effectiveness when facing internal or external changes. By offering the possibility to host multiple PSTs and switch among them on demand during the execution of the system, AIR allows for (self-)adaptation of the system to the

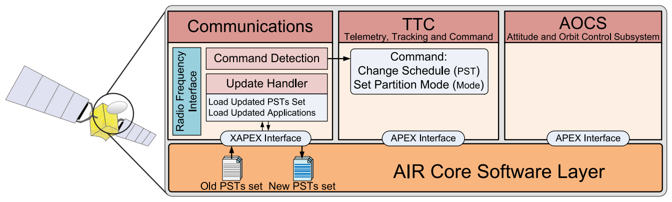


Fig. 3. Spacecraft computing platform

mission's different phases and to detected operational condition changes [10]. Adaptation to current operational conditions is performed exploiting the existing resources and it may not imply the use of update procedures. Nevertheless, we can take profit from update facilities to improve the current features in the spacecraft and manage/optimize resource utilization.

C. Achieving Survivability

Several failures may lead spacecraft to unpredictable and severe conditions which could put the mission at risk. For example, the loss of a gyroscope in a satellite could seriously affect its attitude stability. However, industry practice has demonstrated that recovery from these severe failures can be achieved through the update of software components [4]. In this context, the update of software components and the corresponding configuration procedures play a fundamental role in the survivability of the system.

The build and integration process may benefit from development tools allowing to verify, validate, test and simulate system operation [12]. This includes verifying that real-time guarantees are kept; for this purpose, scheduling tools like Cheddar [8], [9] can allow to quickly and easily define, test and simulate new schedules.

V. Onboard Software Update

We define a methodology to allow the inclusion of new features on a spacecraft during a mission, maintaining the real-time and safety guarantees of the original mission. This should not affect the correct overall behaviour of the system [11]. With the ability to update software components [13], [14], [15], applied to space missions, we can maximize the spacecraft's lifetime and the survivability of the whole mission. The methodology herein defined is currently under implementation.

A. Defining requirements and components

We assume that the upload of the modified software components is supported by a (secure) communication channel and data communication protocol. The data sent by the ground station are received by the system partition associated to communication functions, which is responsible for (i) the identification of the components to be updated; (ii) the allocation of the required memory resources, and; (iii) the functional integration of each component. The partition running the communication functions is assigned a guaranteed processing time. However, we assume that the update operations are performed in a best-effort basis, thus minimizing the impact in the timeliness of the remaining communication functions.

TABLE I
Extended APEX services for Onboard Software Update

Primitive	Short description
XAPEX_MALLOC	Allocate memory from the partition's free memory pool
XAPEX_MFREE	Deallocate a memory zone for the partition's free memory pool
XAPEX_MCLAIM	Claim memory from a specified partition for the partition's free memory pool
XAPEX_PUPDATE	Apply partition software components update
XAPEX_PSTUPDATE	Apply system partition scheduling table (PST) set update

To support the introduction of onboard software update operations, the original APEX interface should be extended to cope with appropriate services, presented in Table I.

B. Integration on spacecraft onboard platform

A typical spacecraft hosts several subsystems, consisting of avionics functions and payload, which closely interact with each other. Relevant examples of avionics functions are the Attitude and Orbit Control Subsystem (AOCS), Communications, Data Handling, and Telemetry, Tracking and Command (TTC). In AIR, these functions are hosted in different partitions and share the same computational platform.

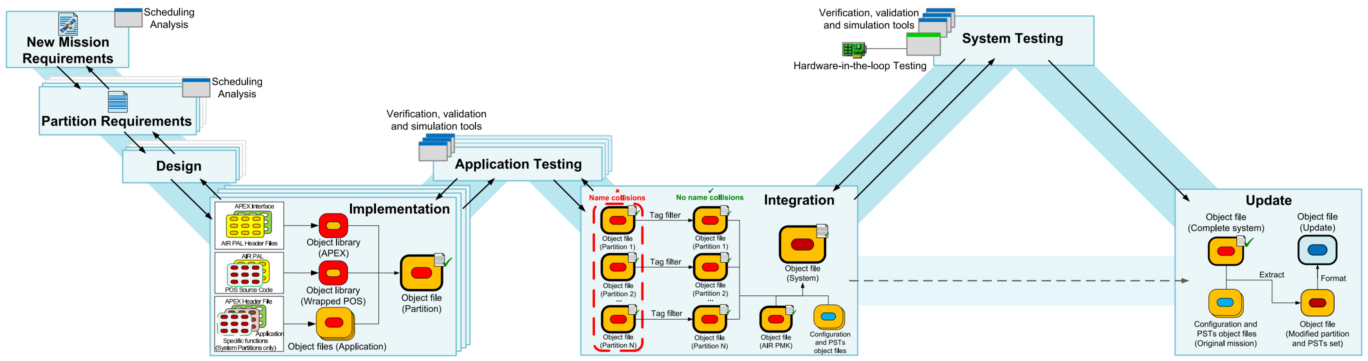
The update is performed by the Update Handler, defined as a process/thread integrated in the partition responsible for the communications. This partition also includes a component for command detection, which will pass the commands issued from the ground station to the TTC through an interpartition communication channel (Fig. 3).

C. Methodology for onboard update

The defined onboard software update methodology exploits the properties provided by the AIR architecture, namely with respect to the composability inherent to the build and integration process. This may consist of the modification of application software, systemwide configurations or simply the definition of new PSTs, in order to upgrade the original mission adapting it according to the new requirements. This consists of a four-step procedure as follows:

1) Offline verification and validation of software modifications: This step corresponds to the AIR original verification and validation process of software components to ensure that safety and timeliness would not be compromised with the introduction of new components. Due to the composability properties, this may be done by the software development teams or providers independently.

2) Extraction of updated components: The final goal of this step is to identify which components need to be uploaded to the spacecraft onboard computer, extract them from the complete system object file, and create a new one composed only of the modified components. Using appropriate tools, this object file should be built according to a specific format in



New Mission Requirements
Scheduling Analysis
Partition Requirements
Scheduling Analysis
Design
Verification, validation and simulation tools
Application Testing
Implementation
APEX Interface
AIR PAL Header Files
AIR PAL
POS Source Code
APEX Header File
Application Specific functions (System Partitions only)
Object library (APEX)
Object library (Wrapped POS)
Object files (Application)
Object file (Partition)
Verification, validation and simulation tools
Hardware-in-the-loop Testing
System Testing
Name collisions
No name collisions
Integration
Object file (Partition 1)
Object file (Partition 2)
Object file (Partition N)
Tag filter
Tag filter
Tag filter
Object file (Partition 1)
Object file (Partition 2)
Object file (Partition N)
Object file (System)
Object file (AIR PMK)
Configuration and PSTs object files
Update
Object file (Complete system)
Object file (Update)
Extract
Format
Configuration and PSTs object files (Original mission)
Object file (Modified partition and PSTs set)

Fig. 4. Impact of the update methodology in the software development life cycle

order to be recognized by the Update Handler. After this step, the new object file will be uploaded to the spacecraft.

3) Transfer of updated components: In the spacecraft, the application and PSTs are received by the partition hosting the communication functions. The Update Handler inspects the uploaded object and separates the application from the PSTs, invoking the XAPEX primitives to (i) allocate and claim memory resources; (ii) assign the updated software components to the specific partition, and; (iii) issue a request to apply the PST update.

4) Activation of updated components: The activation of new PSTs can be performed as follows. The first condition to the safe application of a new set of PSTs is that the currently selected schedule is identical in both the existing and the updated PSTs set. The second condition is that a schedule switch to a PST which has been modified in the updated set is not pending. After the new PSTs have been activated, an uploaded application will be activated after selecting an appropriate schedule.

D. Impact of the methodology in the software life cycle

The methodology defined for updating software components and PSTs introduces new constraints in the software life cycle, with major impact in the verification and validation phases and introducing new steps, namely those related to the extraction of the modified components that will be transferred later to the spacecraft's onboard computing platform. The impact of the methodology is illustrated in Fig. 4, where only the partitions related to new mission requirements need to be modified. The model presented in Fig. 4 represents requirement analysis, design and implementation, followed by the extraction and format of the components that are being updated.

VI. Conclusion and Future Work

In this paper, we overviewed the properties achieved by the AIR architecture and how they can be exploited to reach adaptive time- and space-partitioned systems. Adaptation to changing and unexpected environmental conditions is of paramount importance, since it helps assure the mission's goals. Adaptation to severe incidents or to internal spacecraft failures may take advantage of onboard software update to prolong the lifetime of the spacecraft. Motivated by the need to add new functions to the spacecraft during a mission or to change the mission plans, we defined the requirements and established a methodology for onboard software update, taking advantage of the composability properties inherent to the build and integration process of AIR-based systems.

Future developments of the AIR Technology involve the update of critical software components without interrupting the system execution, remote system monitoring and modification of systemwide control parameters. Issues concerning the spacecraft interaction with surrounding environment through the use of sensors and actuators remain an important requisite.

References

[1] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms and assurance," SRI International, California, USA, Tech. Rep. NASA CR-1999-209347, Jun. 1999.

[2] TSP Working Group, "Avionics time and space partitioning user needs," Technical Note TEC-SW/09-247/JW, Aug. 2009, ESA–ESTEC.

[3] J. Rufino, J. Craveiro, and P. Verissimo, "Architecting robustness and timeliness in a new generation of aerospace systems," in *Architecting Dependable Systems VII*, ser. LNCS, A. Casimiro, R. de Lemos, and C. Gacek, Eds. Springer, 2010, vol. 6420, pp. 146–170.

[4] M. Tafazoli, "A study of on-orbit spacecraft failures," *Acta Astronautica*, vol. 64, no. 2-3, pp. 195–205, 2009.

[5] AEEC (Airlines Electronic Engineering Committee), "Avionics application software standard interface, part 1 - required services," Aeronautical Radio, Inc., ARINC Spec. 653P1-2, Mar. 2006.

[6] ——, "Avionics application software standard interface, part 2 - extended services," Aeronautical Radio, Inc., ARINC Spec. 653P2-1, Dec. 2008.

[7] A. J. Kornecki and J. Zalewski, "Certification of software for real-time safety-critical systems: state of the art," *ISSE*, vol. 5, no. 2, 2009.

[8] J. Craveiro and J. Rufino, "Schedulability analysis in partitioned systems for aerospace avionics," in *Proc. of the 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, Bilbao, Spain, Sep. 2010.

[9] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, no. 4, 2004.

[10] J. Craveiro and J. Rufino, "Adaptability support in time- and space-partitioned aerospace systems," in *Proc. 2nd Int. Conf. on Adaptive and Self-adaptive Systems and Applications*, Lisbon, Portugal, Nov. 2010.

[11] J. Rosa, J. Craveiro, and J. Rufino, "Exploiting AIR composability towards spacecraft onboard software update," in *Actas do INForum - Simpósio de Informática 2010*, Braga, Portugal, Sep. 2010.

[12] A. T. Bahill and S. J. Henderson, "Requirements development, verification, and validation exhibited in famous failures," *Systems Engineering*, vol. 8, no. 1, pp. 1–14, 2005.

[13] M. Neukirchner, S. Stein, H. Schrom, and R. Ernst, "A software update service with self-protection capabilities," in *DATE*, 2010, pp. 903–908.

[14] M. Hicks, "Dynamic software updating," *ACM Transactions on Programming Languages and Systems*, vol. 27, no. 6, Nov. 2005.

[15] M. Wahler, S. Ritcher, and M. Oriol, "Dynamic software updates for real-time systems," in *Proc. HotSWUp'09*, Orlando, FL, USA, Oct. 2009.