Exploiting AIR Composability towards Spacecraft Onboard Software Update

Joaquim Rosa, João Craveiro, and José Rufino

[*] Universidade de Lisboa, Faculdade de Ciências, LaSIGE

Abstract. The AIR architecture, developed to meet the interests of the aerospace industry, defines a partitioned environment for the development of aerospace applications, adopting the temporal and spatial partitioning (TSP) approach, and addressing real-time and safety issues. The AIR Technology includes the support for mode-based schedules, allowing to alternate between scheduling modes during a mission, according to different mission's operation plans. Furthermore, it can be necessary, useful or even primordial having the possibility to host new applications in the unmanned spacecraft onboard computer platform in execution time. In this paper we define the foundations of a methodology for onboard software update, taking advantage of the composability properties of the AIR architecture, in order to add new features to the mission plan.

Resumo. A arquitectura AIR, desenvolvida para responder aos interesses da indústria aeroespacial, define um ambiente compartimentado para o desenvolvimento de aplicações aeroespaciais que adoptem a abordagem de compartimentação temporal e espacial, discutindo questões de tempo-real e de segurança no funcionamento. A Tecnologia AIR inclui o suporte para alternar entre vários modos de escalonamento durante uma missão, de acordo com diferentes planos de funcionamento. Além disso, pode ser necessário, útil ou mesmo primordial ter a possibilidade de alojar novas aplicações ou funcionalidades no computador de bordo do veículo espacial não-tripulado em tempo de execução. Neste artigo definimos os fundamentos de uma metodologia para actualização de software durante o funcionamento do sistema, aproveitando as propriedades de componibilidade da arquitectura AIR, para adicionar novas funcionalidades ao plano da missão.

1 Introduction

Future space missions aiming long-term durations call for a new generation of spacecrafts. This has driven the interest from the space agencies and industry partners in the definition and design of fundamental building blocks for onboard computer platforms, where the strict demands for reliability, timeliness, safety and security are combined with an overall requirement to reduce the size, weight and power consumption (SWaP) of the computational infrastructure.

[*] 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.

The definition of partitioned architectures implementing the logical containment of applications in criticality domains, named partitions, allows to host different applications in the same computational infrastructure and enables the fulfilment of those requirements [14]. The notion of temporal and spatial partitioning (TSP) ensures that the activities in one partition do not affect the timing of activities in other partitions and prevents the applications to access the addressing space of each other.

The AIR (ARINC 653 In Space Real-Time Operating System) Technology emerges as a partitioned architecture for aerospace applications [13] applying the TSP concepts. The AIR architecture allows the execution of both real-time and generic operating systems in independent partitions, ensures independence from the processing infrastructure and enables independent verification and validation of software components.

In a partitioned architecture, the several functions of an unmanned spacecraft, such as Attitude and Orbit Control Subsystem (AOCS), Telemetry, Tracking and Command (TTC) subsystem, share the same computational resources, being hosted in different partitions. Partitions are scheduled according to fixed cyclic scheduling tables. The AIR architecture allows the possibility to dynamically alternate between different scheduling tables. This is useful for the adaptation of partition scheduling to different mission operating modes and for the accommodation of component failures [13].

During the course of a mission, situations may appear on which it may be useful or even necessary to introduce new functions or to modify existing ones to deal with unexpected events. For example, in the presence of a failure of a specific component, it may be necessary to change the mission plan by reconfiguring the applications' scheduling. An example where such features had an important role was the incident with NASA's rover Spirit [4]. In May 2009 the rover was stuck on Mars soft sand terrain and after some months of trying to release it without success, the NASA's team decided to change the mission plan and instead of doing surface exploration, the rover started working as a stationary research platform, performing functions that would not be possible to a mobile platform, such as detecting oscillations in the planet's rotation which would indicate a liquid core.

The modular design of the AIR architecture and the separation of applications in the temporal and spatial domains enables composability properties which are exploited in the build and integration process. This means that the several components can be developed, verified and validated independently. To a software provider, this procedure does not depend on knowledge of the other partitions and, at most, is aided by guidelines to accomplish timeliness requirements. To the system integrator, it is assigned the responsibility of ensuring the accomplishment of system-wide temporal requirements. This paper addresses how to take advantage of the composability properties of the AIR architecture to establish the basis of an onboard software update methodology.

The remainder of this paper is organized as follows. In Section 2 we describe the AIR Technology including the schedulability and composability properties of the architecture, and the build and integration process. In Section 3, we describe the requirements, the components and the integration process of the onboard software update, along with the methodology defined. In Section 4, we expose future research directions and some related work. Finally, Section 5 concludes the paper.

2 AIR Technology Design

The AIR Technology design was original prompted by the interest of the European Space Agency (ESA) in the adoption of TSP concepts to the space industry. The AIR Technology is currently evolving towards an industrial product definition by improving and completing its architecture definition and engineering process [12,13].

2.1 System Architecture

The AIR architecture, illustrated in Fig. 1, allows applications to be executed in logical containers called partitions. At the application software layer (Fig. 1) applications consist in general of one or more processes, which make use of the services provided by an *Application Executive* (APEX) *interface*, as defined in ARINC 653 specification [1]. In addition, a system partition may invoke also specific functions provided by the core software layer, thus being allowed to bypass the standard APEX interface (Fig. 1).

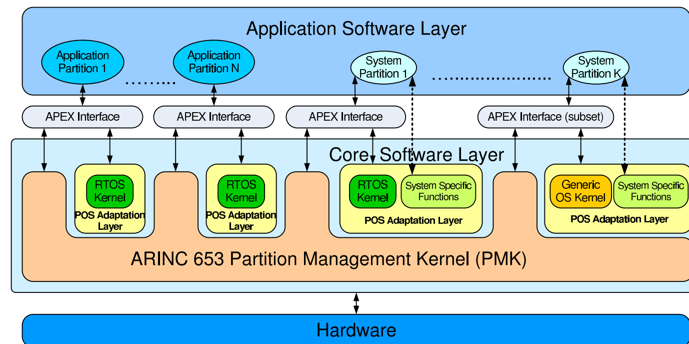


Fig. 1. AIR Architecture and Integration of Partition Operating Systems

The core software layer provides a (real-time or generic) operating system kernel per partition herein referred to as *Partition Operating System* (POS). The *AIR POS Adaptation Layer* (PAL) [6] wraps each POS, hiding its particularities from the AIR architecture components.

The AIR architecture implements the advanced notion of portable APEX, meaning portability between the different POSs is built on the availability of PAL related functions and on the APEX core layer, which may exploit the POSIX application programming interface available on most (real-time) operating systems. The APEX provides the required partition and process management services, time management services, intra-partition and inter-partition communication services and health monitoring services.

The partition management, inter-partition communication and health monitoring services rely additionally on the *AIR Partition Management Kernel* (PMK) service interface. The AIR PMK bears the most responsibility in ensuring robust TSP. The temporal partitioning is achieved by scheduling the partitions according to a given scheduling

table, repeated cyclically over a *major time frame* (MTF). The spatial partitioning is ensured by a high-level abstraction layer which provides a mapping between AIR protection requirements and the hardware's addressing space protection mechanisms.

The AIR architecture also incorporates a *Health Monitor* (HM) component to handle hardware and software errors, containing them within their domains of occurrence.

2.2 Temporal and Spatial Partitioning

To ensure the safety and timeliness of mission-critical systems and minimize the drawbacks arising from the integration of multiple functions sharing the same hardware resources, the design of AIR Technology proposes the architectural principle of robust partitioning. With partitioning we achieve two important properties. The first concerns containing the occurrence of faults to the context where they appear, and thus not interfering with the system overall behaviour. The other property has to do with system *composability* enabling the independent verification and validation of software components that also facilitates the overall certification process, fundamental for space-borne vehicles.

The AIR architecture has been designed to fulfil the requirements for robust TSP. Temporal partitioning ensures that the activities processed in one partition do not affect the real-time requisites of the functions running in other partition. Space partitioning relies on having separate addressing spaces and thus not allowing an application to access the memory and input/output (I/O) spaces of a different partition.

2.3 Designing for Schedulability

The original ARINC 653 [1] notion of a single fixed partition scheduling table, defined offline, is limited in terms of timeliness control and fault tolerance. The design of the AIR architecture incorporates the advanced notion of *mode-based partition schedules*, allowing temporal requirements to vary according to the mission's phase or mode of operation [13,2].

An AIR-based system includes a set of partition schedules among which it can switch during its operation. A schedule switch can be ordered by a specific partition designed and allowed to do so, through the invocation of an APEX primitive. This can, in turn, result from either a command issued from ground control or from reacting to environmental conditions as obtained by the spacecraft's sensors. The order will not come into immediate effect, but rather applied at the end of the current MTF.

The AIR Partition Scheduler component is responsible for guaranteeing that the processing resources are, at every time, assigned to the correct partition and for making schedule switch effective at the end of the respective MTF. Its implementation is described in pseudocode in Algorithm 1. This is executed at every system clock tick, inside the respective interrupt service routine. The implementation of this algorithm is optimized to introduce little overhead to such routine.

The first verification to be made is whether the current instant is a partition preemption point (line 2). In case it is not, the execution of the partition scheduler is over; this is both the best case and the most frequent one. If it is a partition preemption point, we

Algorithm 1 AIR Partition Scheduler featuring mode-based schedules

1: $ticks \leftarrow ticks + 1$ ▷ $ticks$ is the global system clock tick counter
2: **if** $schedules_{currentSchedule}.table_{tableIterator}.tick =$
$(ticks - lastScheduleSwitch) \bmod schedules_{currentSchedule}.mtf$ **then**
3: **if** $currentSchedule \neq nextSchedule \wedge$
$(ticks - lastScheduleSwitch) \bmod schedules_{currentSchedule}.mtf = 0$ **then**
4: $currentSchedule \leftarrow nextSchedule$
5: $lastScheduleSwitch \leftarrow ticks$
6: $tableIterator \leftarrow 0$
7: **end if**
8: $heirPartition \leftarrow schedules_{currentSchedule}.table_{tableIterator}.partition$
9: $tableIterator \leftarrow (tableIterator + 1) \bmod$
$schedules_{currentSchedule}.numberPartitionPreemptionPoints$
10: **end if**

then verify (line 3) if there is a pending scheduling switch to be applied and if the current instant is also the end of the MTF. If these conditions apply, then a different partition scheduling table will be used henceforth (line 4). The partition which will hold the processing resources until the next preemption point, dubbed the heir partition, is obtained from the partition scheduling table in use (line 8) and the AIR Partition Scheduler will now be set to expect the next partition preemption point (line 9).

2.4 Designing for Composability

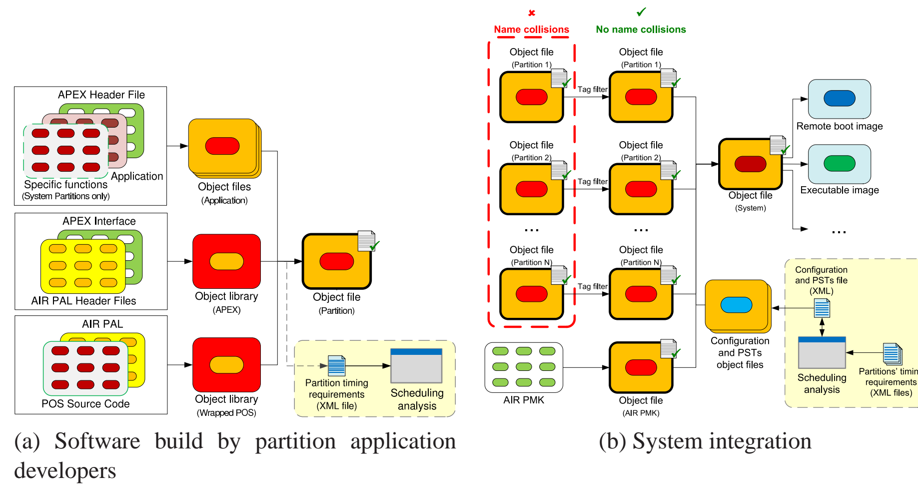
The design of the AIR architecture and the use of a TSP approach enables the *composability properties* of AIR-based systems, in both time and space domains. The use of a fixed cyclic partition scheduling scheme dictates that the timeliness guarantees of each partition are defined by the processing time assigned to each partition. In the spatial domain the composability properties ensure that the partition's memory and I/O resources are protected against unauthorized access from other partitions. The composability properties are thus inherent the AIR modular architecture.

The modularity of the AIR architecture design and of its build and integration process further enables the composability of AIR-based systems [5]. This means, on a first approach, that the several components that may compose such a system can be developed, verified and validated independently. This eases certification efforts, since only modified modules need to be reevaluated. It is also a fundamental basis for onboard software update as proposed in this paper.

From the point of view of one partition's provider, this further signifies that development and validation does not depend on knowledge of the other partitions (individually or as a whole). At most, the development of one partition should be aided by a set of guidelines for its applicability to the target TSP systems in general. The system integrator is responsible for guaranteeing a correct partition scheduling, so that partitions and the system as a whole meet their timing requisites [5].

2.5 Build and Integration Process

Because of the particularities of the architecture, the software build and integration process needs to differ from the canonical application build process, as provided by standard compilers and linkers. This process is pictured in Fig. 2 and it will now be described in detail.



(a) Software build by partition application developers

(b) System integration

Fig. 2. AIR build and integration process

Partition build process

The first stage concerns building each partition independently (Fig. 2a). In the typical scenario, the applications to be executed in the context of a partition, the APEX library, and the underlying POS libraries (wrapped by the AIR PAL) may be provided by different teams or providers. Therefore, the build process is tailored to expect these independent object files, and link them together to produce an object file with no unresolved symbols but including relocation information (to allow linking with the remaining partitions). Although the AIR PAL also invokes the AIR PMK (which symbols are as of yet undefined), these interactions are wrapped using data structures to reference the appropriate primitives, which the AIR PMK will register by executing code generated at system integration time with the assistance of a specific AIR tool.

The introduction of a scheduling analysis phase in the application developers' software production chain [5] takes advantage of the composability properties to provide independent schedulability analysis. Application developers can perform this analysis using the timing requirements (period, worst-case execution time, deadline, etc.) of their applications' processes. This information can be either estimated, or tentatively determined through static code analysis [11].

System integration

The system integration process (Fig. 2b) receives input (partition object files) from potentially different teams or providers. Since all partitions will include the common interface provided by the AIR PAL and AIR APEX libraries, the various partitions' object files will have symbol name collisions; partitions running the same POS or POSs providing the same standardized interfaces (e. g., POSIX) have additional name collisions. Therefore, linking these objects will require previous preprocessing. This preprocessing can be in the form of a *tag filter* utility which prefixes all symbols and calls in each partition's object files with unique prefixes (e. g., `P1`, `P2`, etc.). This process can be further optimized by automating the generation of partition prefixes, namely deriving them from the configuration file.

The partition objects can now be linked with the AIR PMK and the configuration object. This configuration object is derived by compiling C source code files, which in turn have been converted from XML (Extensible Markup Language) configuration files. The use of XML for the configuration file is motivated by the overall intention to comply, up to a certain degree, with the ARINC 653 specification [1]. Besides the parameters translated from these XML files (such as partition scheduling tables, addressing spaces, and inter-partition communication ports and channels), configuration objects include routines for the AIR PMK to register the adequate primitives in the AIR PAL structures. This linking step produces the system object file, from which in turn one can generate the most adequate deployment format for the target platform. In the system integration phase, scheduling analysis capabilities shall be introduced in relation with the generation of a system-wide configuration [5].

3 Onboard Software Update

We establish the foundations of a methodology to allow including new features on a spacecraft during a mission. The challenges we face are related to maintaining the real-time and safety guarantees defined for the original mission. Adding a new application to the system should be performed in a way that does not affect the overall behaviour of the system, including the timeliness of the already running applications.

3.1 Defining Requirements and Components

To support the upload of modified software components to the spacecraft's onboard computer platform, we assume the existence of a (secure) communication channel and a data communication protocol. The communication functions aboard the spacecraft are responsible for dealing with the reception of the data sent by the ground station and for performing online processing of the transferred data stream. Handling the update of onboard software components implies: the identification of the components being updated (partition software components, PSTs sets); the allocation of the required memory resources; the functional integration of each component in the operation of the onboard computer platform. The onboard software update handler shall be implemented as an activity (process/thread) in the domain of the (system) partition associated to the communication functions.

To the partition hosting the communication functions it is ensured a given time processing budget. However, we assume that software update activities are performed on a best-effort basis, thus with minimal impact on the timeliness of the communication functions. This ensures the safety of onboard software update since it will not interfere with other communication functions, namely with the detection and the identification of ground commands.

To support the introduction of onboard software update operations, the original APEX interface must be extended with the services presented in Table 1. However, only the APEX interface of the partition hosting the onboard software update functions needs to be extended.

Table 1. 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

3.2 Integration on Spacecraft Onboard Platform

We assume the component dedicated to onboard software update, the Update Handler, is defined as a process/thread integrated in the partition responsible for the communication functions, as illustrated in the simplified spacecraft architecture [8], pictured in Fig. 3. This partition also includes a command detection function. Commands issued from ground mission control will be passed to the TTC through a inter-partition communication channel. One example is a ground command to change a PST.

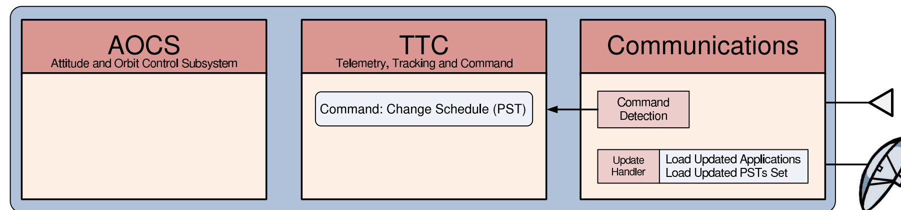


Fig. 3. Spacecraft onboard platform

3.3 Designing an Onboard Software Update Methodology

The design of a methodology for onboard software update in AIR-based systems has evolved from the build and integration process. This methodology is extended to cope with the modification of software components in order to upgrade the original mission.

This may include the modification of application software, partition or system wide configurations or simple the definition of a new set of partition scheduling tables (PSTs). The complete methodology consists in a four-step procedure as follows:

STEP 1: Offline Verification and Validation of Software Modifications

The modifications to the software components of a given mission may include the redesign of the applications associated with a given partition (e.g., payload functions) and the definition of a new set of PSTs. The linking of the modified partition with the objects of other partitions is made on the logical address space in order to guarantee that the mapping of unmodified partitions remains unchanged. This way, only the updated components need to be uploaded to the spacecraft onboard computer platform. This process is illustrated at the left side of Fig. 4 and may involve scheduling analysis of the partition. The update of the mission may simply involve the modification of a given PSTs set. In this case, the schedulability analysis and the generation of a new configuration and PSTs set is only performed at the system integration stage.

This corresponds to the AIR original verification and validation process of software components performed on the ground, before sending the applications to the spacecraft, and consists on applying the build and integration process to ensure that the safety and the TSP requirements would not be compromised with the introduction of new components on the system. Due to the composability properties of the AIR architecture, the build process may be done by the software development teams or providers independently. Each team or provider, along with the new application, delivers the partition timing requirements, that altogether will form the partition scheduling tables (PSTs), used by the AIR PMK Partition Scheduler on the target system.

The output produced in this step is the system object file, resulted from the integration of all the built objects potentially from various developers.

STEP 2: Extraction of Updated Components

After having the result of the build and integration process done on the previous step, there is the need to identify which components need to be uploaded to the spacecraft onboard computer platform. The final goal of this step is to identify those components, extract them from the complete system object file and create a new object composed only by the components to be uploaded to the spacecraft onboard computer. Also, it is necessary to build the object file according to a specific format, in order to the Update Handler be able to recognize the data received and perform its handling.

Like the previous one, this step is made on the ground. It requires a special-purpose toolset to perform the extraction and the formatting functions. The extraction and the formatting actions are represented by the shaded area at the right side of Fig. 4.

Finally, the updated object will be uploaded to the spacecraft using the communication facilities to exchange data between the ground stations and the space vehicles.

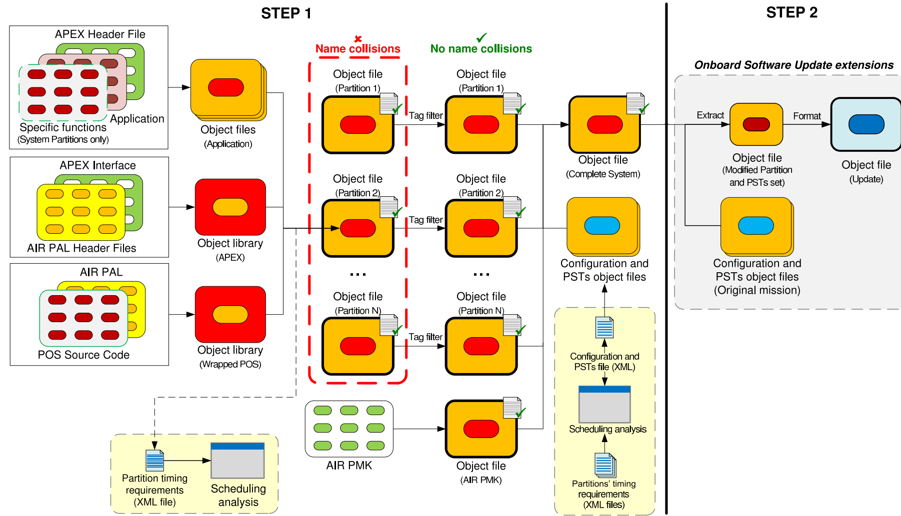


Fig. 4. Integration of an AIR-based system extended with the extraction and formatting of the updated components

STEP 3: Transfer of Updated Components

In the spacecraft, the application and PSTs uploaded in a single object file are received by the partition running the application responsible for the communication operations. Complementarily to the formatting done in the Step 2, when the modified components were formatted into an object file, the Update Handler look into the uploaded object file and separate the application of the PSTs.

We assume the existence of a component which will provide the required communication facilities between the spacecraft and the ground stations.

Upon reception of partition software components the Update Handler will invoke the XAPEX_MALLOC primitive to allocate the required memory. We assume that the available memory is large enough to contain the updated application. The Update Handler may also invoke the XAPEX_MCLAIM primitive to claim the memory used by the partition being updated, followed by the XAPEX_PUPDATE primitive which assigns the updated software components to the specified partition (Table 1). Finally, upon reception of a PSTs set, the Update Handler will invoke the XAPEX_PSTUPDATE primitive which will apply the PST set update.

STEP 4: Activation of Updated Components

To guarantee that applying the updated PSTs set does not compromise the safety of the whole mission, the XAPEX_PSTUPDATE (Table 1) will perform a blocking wait until the proper conditions are met, as described in Algorithm 2. The first condition for safe application of a new set of PSTs is that the currently selected schedule is identical in both the existing and the updated PSTs sets. The second condition is that a schedule

switch to a PST which has been modified in the updated set is not pending. The goal of these conditions is to ensure that the operation conditions that the system expects and/or the criteria by which the system or a ground operator has chosen the current or next schedule are not voided.

Algorithm 2 XAPEX_PSTUPDATE primitive

1: **while** $schedules_{currentSchedule} \not\equiv newSchedules_{currentSchedule} \lor$
$schedules_{nextSchedule} \not\equiv newSchedules_{nextSchedule}$ **do** ▷ Wait (block)
2: **end while**
3: SWAP($schedules$,$newSchedules$)

After the new PSTs have been activated, the uploaded partition application can now be scheduled, a situation which may occur upon receiving a schedule switch command from the ground mission control, as illustrated in Fig. 3.

4 Future Developments and Related Work

The importance of a strong verification and validation process in critical systems is addressed in [3] and the relevance of a safety-policy validation at binary level is highlighted in [10]. The problem of dependable online upgrade of real-time software was approached in [16].

The methodology established in this paper for onboard software update can be further extended to cope with the upgrade of critical software components that must be performed without interruption, such as those ensuring AOCS, TTC and communication functions. This implies a new set of challenges to be addressed specifically in the steps 3 (transfer of updated components) and 4 (activation of updated components). Although driven by the specific requirements of aerospace applications, these developments may benefit from the work performed on dynamic software update [7,9,15,17].

Solutions for dynamic software update on real-time systems requiring the identification of specific points in time for components' update is discussed in [17], while [15] makes no presumption about new application's periods and execution times.

The results achieved in [7] shown that the real-time and fault-tolerant requirements of avionics systems could be accomplish even during a dynamic reconfiguration of the system due to component failures. An approach for dynamic update of applications in C-like languages is provided in [9] and focuses on the update of the code and data at predetermined times, but does not specify real-time requirements.

5 Conclusion

In this paper we described the AIR Technology, towards build aerospace applications to temporal and spatial partitioning systems. Motivated by the need to add new applications in the system during a mission, due to changing its plans, we defined the onboard software update requirements and discussed how to take advantage of the composability

inherent to the build and integration process of the AIR-based systems. We establish a methodology for onboard software update, that exploits the composability properties of the AIR architecture, allowing independent verification and validation. The onboard software update methodology is based on the redefinition of the original space mission and it is supported on a specific toolset for the extraction of the updated software components, to be uploaded to the spacecraft onboard computer. The methodology can be further extended to support dynamic update of critical software components.

References

1. AEEC (Airlines Electronic Engineering Committee): Avionics application software standard interface, part 1 - required services. ARINC Specification 653P1-2 (Mar 2006)
2. AEEC (Airlines Electronic Engineering Committee): Avionics application software standard interface, part 2 - extended services. ARINC Specification 653P2-1 (Dec 2008)
3. Bahill, A.T., Henderson, S.J.: Requirements development, verification, and validation exhibited in famous failures. Systems Engineering 8(1), 1–14 (2005)
4. Brown, D., Webster, G.: Now a Stationary Research Platform, NASA's Mars Rover Spirit Starts a New Chapter in Red Planet Scientific Studies. http://www.nasa.gov/mission_pages/mer/news/mer20100126.html (Jan 2010)
5. Craveiro, J., Rufino, J.: Schedulability analysis in partitioned systems for aerospace avionics. In: Proc. 15th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA 2010). Bilbao, Spain (Sep 2010)
6. Craveiro, J., Rufino, J., Schoofs, T., Windsor, J.: Flexible operating system integration in partitioned aerospace systems. In: Actas do INForum - Simpósio de Informática 2009. Lisboa, Portugal (Sep 2009)
7. Ellis, S.M.: Dynamic software reconfiguration for fault-tolerant real-time avionic systems. Microprocessors and Microsystems 21, 29–39 (1997)
8. Fortescue, P.W., Stark, J.P.W., Swinerd, G. (eds.): Spacecraft Systems Engineering, 3rd Edition. Wiley (2003)
9. Hicks, M.: Dynamic software updating. ACM Transactions on Programming Languages and Systems 27(6), 1049–1096 (Nov 2005)
10. Necula, G.C., Lee, P.: Safe kernel extensions without run-time checking. In: Proc. USENIX 2nd Symposium on Operating Systems Design and Implementation. pp. 28–31 (1996)
11. Pushner, P., Koza, C.: Calculating the maximum execution time of real-time programs. Journal of Real-Time Systems 1, 160–176 (Sep 1989)
12. Rufino, J., Craveiro, J., Schoofs, T., Tatibana, C., Windsor, J.: AIR Technology: a step towards ARINC 653 in space. In: Proceedings of the DASIA 2009 "DAta Systems In Aerospace" Conference. EUROSPACE, Istanbul, Turkey (May 2009)
13. Rufino, J., Craveiro, J., Verissimo, P.: Architecting robustness and timeliness in a new generation of aerospace systems. In: Casimiro, A., de Lemos, R., Gacek, C. (eds.) Architecting Dependable Systems 7. LNCS, Springer, Berlin Heidelberg (2010), accepted for publication
14. Rushby, J.: Partitioning in avionics architectures: Requirements, mechanisms and assurance. Tech. Rep. NASA CR-1999-209347, SRI International, California, USA (Jun 1999)
15. Seifzadeh, H., Kazem, A., Kargahi, M., Movaghar, A.: A method for dynamic software updating in real-time systems. In: Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science. Shanghai, PR China (Jun 2009)
16. Sha, L.: Dependable system upgrade. In: RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium. p. 440. IEEE Computer Society, Washington, DC, USA (1998)
17. Wahler, M., Ritcher, S., Oriol, M.: Dynamic software updates for real-time systems. In: Proceedings of the HotSWUp'09. Orlando, Florida, USA (Oct 2009)