# EXPERIMENTING MAINTENANCE OF FLIGHT SOFTWARE IN AN INTEGRATED MODULAR AVIONICS FOR SPACE

**Johan Hardy[(1)], Thomas Laroche[(2)], Philippe Creten[(3)], Paul Parisis[(4)], Martin Hiller[(5)]**

[(1)] Spacebel, Liège Science Park, Rue des Chasseurs Ardennais 6, B-4031 Angleur, Belgium, johan.hardy@spacebel.be
[(2)] Spacebel, I. Vandammestraat 5-7, 1560 Hoeilaart, Belgium, thomas.laroche@spacebel.be
[(3)] Spacebel, Liège Science Park, Rue des Chasseurs Ardennais 6, B-4031 Angleur, Belgium, philippe.creten@spacebel.be
[(4)] Spacebel, Liège Science Park, Rue des Chasseurs Ardennais 6, B-4031 Angleur, Belgium, paul.parisis@spacebel.be
[(5)] ESA/ESTEC, Keplerlaan 1, 2200 AG Noordwijk, The Netherlands, martin.hiller@esa.int

## Abstract

*This paper presents an experiment of Flight Software partitioning in an Integrated Modular Avionics for Space (IMA-SP) system. This experiment also tackles the maintenance aspects of IMA-SP systems. The presented case study is PROBA-2 Flight Software. The paper addresses and discusses the following subjects: On-Board Software Maintenance in IMA-SP, boot strategy for Time and Space Partitioning, considerations about the ground segment related to On-Board Software Maintenance in IMA-SP, and architectural impacts of Time and Space Partitioning for PROBA software's. Finally, this paper presents the results and the achievements of the study and it appeals at further perspectives for IMA-SP and Time and Space Partitioning.*

Keywords: *Time and Space Partitioning, Integrated Modular Avionics, Spacecraft, On-Board Software Maintenance, PROBA platform.*

## I. INTRODUCTION

The Integrated Modular Avionics for Space (IMA-SP) is a spin in of the corresponding aeronautic concept into the spacecraft avionics architecture. It aims at managing the increasing amount of mission functions embedded in the On-Board Software (OBSW) while saving on mass, volume and power through higher level of integration and sharing of the computing resources.

In IMA, software applications which were previously running on separate computational nodes in a federated system are now integrated onto the same computational node. In order to still keep these integrated software applications separated from each other, time and space partitioning (TSP, see [1]) is employed. TSP ensures that software applications are separated in time (applications are never running in parallel) and space (application can never access each other's memory).

Over the past few years, space agencies have initiated several research activities in the domain of IMA and TSP to collect the lessons learnt from the aeronautic domain, assess the suitability of the TSP approach for space domain, study the impact of TSP on existing RTOS used in space and perform proof of concept studies.

One major difference between the aeronautical domain and the space domain is the area of software maintenance. While aircraft are serviced regularly, and maintenance can be made with physical access to the system, spacecraft are generally not serviceable, and maintenance does not have physical access to the system.

For software maintenance in aircraft, this can be done during regular maintenance appointments, and complete software images can be downloaded while the aircraft is not operational.

For software maintenance in spacecraft, the only access is via the radio link from the ground station. This link has limited bandwidth so software maintenance is mainly performed by uploading software patches. Uploading complete software images is only done if patching is not possible.

One advantage of a partitioned software system is that the software can be updated in one partition without affecting the software in other partitions.

Furthermore, during software maintenance, the spacecraft will usually have to be in a special mode

where the nominal operation is shut down. In an IMA-based spacecraft, the partitions which are not affected by the software update can, in principle, remain operational during the update of a partition.

One could even imagine that a spacecraft could have "spare" partitions (that is, reserved slots of execution time and memory) which would allow adding new software applications to an existing setup.

Among the initiated studies, this experiment has been managed in the frame of one of these research activities led by the ESA/ESTEC IMA-SP program – namely *In-Flight Hosting of Prototype Applications* project.

The fundamental objective of this experiment is to demonstrate through a representative use case that the successful IMA concepts deployed in the civil aviation industry is also beneficial to Space Avionics and in particular to the OBSW developments and life-cycle. The main objectives are however to explore and to demonstrate approaches for software maintenance of an IMA-SP system.

## II. CASE STUDY

In order to present a representative demonstration, the selection of the case study has been based on existing OBSW developed by Spacebel. Among the candidates, PROBA-2 Flight Software (FSW) has been retained as the best candidate for the experimentation – at the time of the selection, it was the last PROBA satellite launched in space, and PROBA-V was still under development.

PROBA-2 flight software is not a unique choice but it can easily be extended to most of On-Board Software architectures encountered in ESA projects.

In line with the industrial process, the experimentation addresses the spacecraft application partitioning namely for what concerns:

- Definition of the overall IMA-SP system architecture and in particular the scheduling plan, memory allocation and inter-partition communication,
- Development and integration of the AOCS, where the algorithmic part is developed and validated in a model environment, possibly by a specific team or independent industrial company,
- Development and integration of the payload software, where part of the payload management or payload data processing may be developed by a research institute or a university,
- Development and integration of the Central Data Handling System (CDHS),
- Development and integration of the OBSW Maintenance services responsible for the maintenance of the above artefacts.

The partitioning kernel selected for this experiment is XtratuM – product developed by fentISS (fent Innovative Software Solutions). XtratuM is a hypervisor, i.e. a layer of software that provides one or more virtual execution environments for partitions. The XtratuM API and internal operations tends to be compliant with ARINC-653 standard [2].

In a bare hypervisor, and in particular in XtratuM, a partition is a virtual computer rather than a group of strongly isolated processes. When multi-threading (or tasking) support is needed in a partition, then an operating system or a run-time support library has to provide support to the application threads. In fact, it is possible to run a different operating system or none on each XtratuM partition.

The complete IMA-SP system of this experiment, including all the partitions and the hypervisor, has been exercised using the existing PROBA-2 Software Validation Facility (SVF).

The assessment of the study has been performed against the non-regression test scenarios coming from PROBA-2 augmented with specific OBSM test cases.

## III. ON-BOARD SOFTWARE MAINTENANCE IN IMA-SP

In the field of embedded system, the term OBSW Maintenance (OBSM) basically consists in updating whole or parts of the system or application software or data. In an IMA-SP environment, it concerns the partitioning kernel as well as the system and application partitions. The goal is then to be able to update any individual partition independently of the others.

Typical addressed issues are whether the OBSM services can be performed from a central partition or must be distributed in each individual partition; how they can access working or storage memory possibly assigned to other partitions; whether logical or physical, relative or absolute or even virtual addresses must then be used; whether these services are used for direct or deferred applications; whether they can be performed in an atomic way or if partitions must first be put on hold; whether static partition scheduling allows for fast reconfiguration or whether the schedule plan must be temporarily modified; how are functional chains of partitions handled.

Beyond the issues addressed by IMA-SP environment, the boot sequence and the initialisation of a TSP system is probably the most complex. The responsibility of this initialisation may rely on different components, depending on the boot strategy (boot software in PROM, the partitioning kernel, a privileged partition, a secondary boot, etc.). As the boot strategy has implications on the management of the partition images

in volatile (RAM) and non-volatile (PROM, FLASH) memory, it is tightly integrated with the On Board Software Maintenance.

In this sense, the following figure presents the adopted strategy in a pragmatic way but with generic considerations. The adopted approach is to have a dedicated privileged partition responsible for the secondary boot loader. This privileged partition is the OBSM partition.

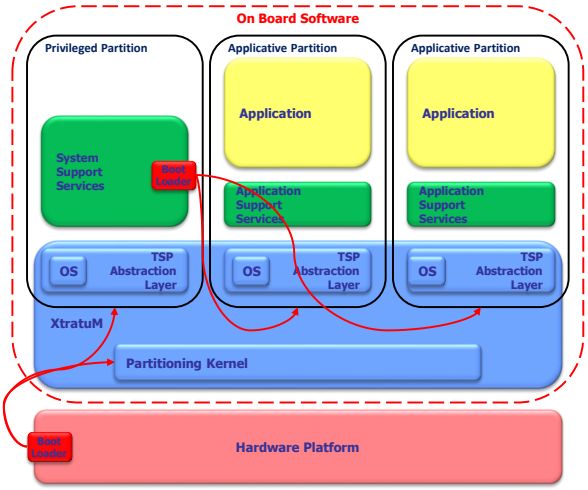


***Figure 1:*** *Selected approach for OBSM and boot strategy*

Although the On-Board Software is the main point of attention, this study puts also its efforts on the ground segment. Indeed, the On-Board Software must rely on a flexible but also reliable ground segment able to manage the complete maintenance process – the process includes the verification of the pre-conditions, the actual maintenance operational commands and the final verification of the maintenance operation.

In order to fulfil this requirement, a prototype of ground tools able to generate adequate sets of commands has been developed. This prototype makes preliminary verifications of the space system state – it checks the coherency of the system w.r.t. the memory organisation, partition type, versions of the artefacts, etc. – and final verifications that the update was performed correctly. This pilot tool fills, in some way, the gap between flight and ground segments.

## IV. PROBA-2 FLIGHT SOFTWARE AND IMA-SP

The PROBA-2 FSW needs to be adapted and modified in order to be able to execute in the chosen IMA-SP environment. In addition, the complete software needs to be adapted to run on LEON3 processor. Most of these aspects have already been considered up to some levels in the previous IMA activities and use cases but for different flight software's and avionics.

Concerning PROBA-2, the software is decomposed in five main functions/partitions:

- The Central Data Handling System partition, which groups the functions such as TM/TC Handling, Data Acquisition and Data Handling, PUS Services – it contains most of the existing PROBA-2 software,
- The AOCS Algorithms partition,
- The Application Software (SWAP) partition, which is the Image Processing partition and Instrument Manager of the SWAP payload. SWAP is a sun-observation instrument,
- Another Application Software partition, which is the second demonstration application that may be integrated within the PROBA-2 software,
- And, finally, the OBSM partition.

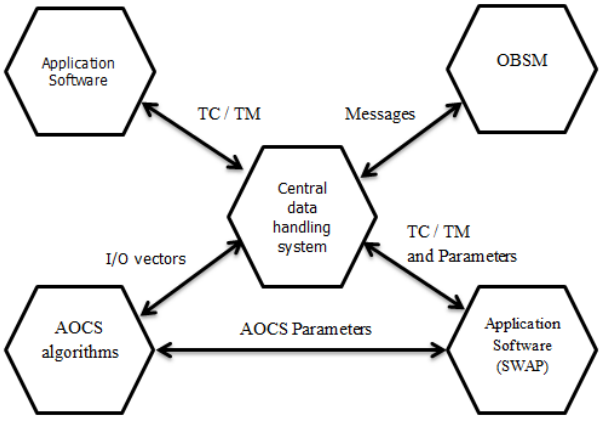


***Figure 2:*** *Partitioned PROBA-2 Flight Software*

This partitioning approach allows focusing the study on the integration of software developed by independent teams and leaves aside the problems of redefining internal interfaces wherever possible.

Besides the process of development, the presented partitioning of the PROBA-2 OBSW has many impacts. Indeed, decomposing into many partitions existing software that has not initially been designed to be partitioned leads to a lot of difficulties not directly related to the study. The following aspects of the OBSW have been revisited in the FSW design:

- PUS Services
- TM/TC Handling
- I/O access
- Scheduling and real-time requirements
- Interrupts/Traps
- Memory/Patch management (cf. OBSM)
- Memory scrubbing

In order to slice PROBA-2 in single parts, we adopted an approach with two iterations to reach a stable partitioned version of the PROBA-2 OBSW:

- A first iteration consisting in having the complete PROBA-2 FSW in a single partition running over the SEP and the SVF, and
- A second iteration, more complex, consisting in separating applications and the main functions into partitions.

Of course, by following this approach, architectural and dynamic aspects generated problems at different levels. The paper puts in evidence the architectural, dynamic and performance issues raised while adapting the existing software to an IMA-SP system.

# V. ASSESSMENT & DISCUSSION

Several further considerations related to the chosen boot strategy are discussed in the next lines, as the boot has many characteristics in common with maintenance: indeed, software maintenance often assumes partition or system reboot. This paragraph discusses the various choices and issues that were raised in the course of this study, and presents the possible approaches and chosen solution.

## V.1 PARTITION BOOT & PARTITION SCHEDULING

The Partition Scheduling impacts the maintenance at two levels: for the boot strategy and for the partition reload.

XtratuM foresees (but does not enforce) at least three scheduling plans: an INIT scheduling plan that is used during system boot (this plan is mandatory and is only used for the boot), a NOMINAL or OPERATIONAL plan to be used during the nominal life of the system (there could be more nominal/operational plans) and a MAINTENANCE plan to be used when performing system maintenance, such as update of partitions.

Concerning the partition boot, in the central staged boot approach, the secondary boot is performed in the frame of the OBSM partition. This implies that the boot time is impacted by the scheduling policy, i.e. by the partition scheduling plan.

The same applies for partition reload (possible both for central staged and distributed boot approach): in a central staged boot approach, the partition reload is performed in the frame of the OBSM partition; in the distributed boot approach, the partition reload is performed in the partition slot context.

There is no solution that would perfectly fit all the needs of all IMA-SP systems. But the proposed boot/reload approach is flexible enough to be able to cope with most situations. Concerning the bootloader strategy, few considerations that should be taken into account when determining the boot approach for a specific project are given below.

About the use of different INIT and OPERATIONAL scheduling plans for the system boot:

- In case a pure central staged boot approach with OBSM partition is selected, it may be interesting to have an INIT scheduling plan providing most of the slots to the OBSM partition, in order to speed up the initialization time. Once the initialization is done, the OBSM partition could request a switch to the OPERATIONAL scheduling plan.
- In case the chosen approach is more distributed, it is arguable whether a different INIT scheduling plan is required or not. In some cases, the OPERATIONAL scheduling plan could be compatible with the time required for booting the partition, implying that INIT and OPERATIONAL scheduling plans could be merged.

In the OBSM solution, the OBSM partition automatically commands a configurable scheduling plan (typically OPERATIONAL) once initialisation is finished (whatever the boot strategy).

About reloading a partition and MAINTENANCE scheduling plan:

- A central staged boot approach with OBSM partition allows reloading a partition; a distributed boot approach where the secondary bootloader is executed from non-volatile memory allows it as well. When reloading and rebooting a partition, it is again questionable whether the OPERATIONAL scheduling plan is adapted. If not, it can be envisaged to use a MAINTENANCE scheduling plan. However, as the whole system is impacted by the scheduling plan, this should be evaluated very carefully[1]. If such a MAINTENANCE scheduling plan is deemed necessary, it could be identical to the INIT scheduling plan.

The OBSM makes no hypothesis on the boot/reload approach. It supports the central staged boot approach, but does not enforce it (i.e. distributed boot remains possible). Similarly, it does not make any hypothesis on the scheduling plans to be used in the various situations that can be encountered in a partitioned system: it allows the ground segment to command the scheduling plan depending on the needs. So, depending on the situation and on the boot/reload strategy, the operator could incorporate a switch of scheduling plan while performing the system maintenance.

1 Since the whole IMA-SP system as well as each partition needs to be qualified for every scheduling plan.

## V.2 MULTI-CORE OR MULTI-PROCESSOR COMPUTER

In the present study, the target computer was a single core LEON3 processor. In the future, partitioned systems should be deployed on multi-core and even multi-processor computer architectures. Such a deployment will have important impacts on the overall system architecture. The notions of scheduling plan and slots have to be extended for instance.

With respect to the IMA-SP system maintenance, it would be interesting to have the possibility to load several partitions in parallel, e.g. in order to decrease the overall boot time. In case the IMA-SP system executes on a multi-core computer, it is reasonable to assume that one partition always executes on the same core (this is the notion of processor affinity). With the centralized boot approach, there is a single OBSM partition in charge of loading the other partitions; moreover this is a sequential process. One solution could be to switch to a distributed boot approach (with secondary boot loader on each partition). Another possibility is to adapt the OBSM partition to take profit of the multi-core/multi-processor capability, e.g. by having several instances of OBSM partitions in parallel (one OBSM partition per core). In that case, it is questionable whether there should be only one OBSM partition handling the maintenance commands from the Central DHS... The access to the shared resources could be a bottle-neck in that case: the advantages of parallel boot of partitions could be partly compensated by the contention on the memory bus due to concurrent access to the non-volatile memory.

However, these aspects have not been considered further in the present study but it appeals at new developments.

## V.3 ON-BOARD SOFTWARE MAINTENANCE GROUND TOOL

The maintenance of flight software implies the interaction of the ground segment with the flight software. The new software to be uploaded is prepared on ground and then uplinked and applied on board. The exact share of responsibility between flight and ground segments results from a trade-off.

The OBSM solution is composed of two parts:

- An OBSM partition executed on a partitioned system, in charge of the initial boot of partitions, and in charge of executing the maintenance commands issued by the Ground.
- An OBSM Ground Tool that manages the different versions and configurations of the partitioned system (hypervisor and partitions). This OBSM Ground Tool is in charge of the generation of the commands for the system maintenance.

The main philosophy of the OBSM solution is to have an OBSM partition in charge of executing rather simple commands, usually writing or reading memory. The OBSM Ground Tool is in charge of ensuring a coherency of the system. This is achieved by keeping a mirror of the flight memory organisation on ground, as well as by organising the command sequences into 3 steps: precondition checks, actual maintenance command and verification checks. So, whenever a new software is uploaded, the on board memory is verified against its expected contents.

The main features of the OBSM Ground Tool are:

- Central repository for the XtratuM image and configuration;
- Central repository for the partition images and configuration;
- Central repository for the overall memory organisation/configuration (including non-volatile memory);
- Management of the “coherent systems” of the repository;
- Management of the “functional dependency” between partition;
- Generation of commands to copy XtratuM image in non-volatile memory;
- Generation of commands to copy partition images and configuration in non-volatile memory;
- Generation of commands to copy partition patches in non-volatile memory;
- Generation of commands to the OBSM partition divided into pre-condition, commands and verification;
- Generation of the OBSM partition configuration file, used for the boot process and maintenance (this configuration could possibly be used by a generic secondary bootloader in case a distributed boot approach is preferred);
- Maintenance of a ground mirror image of the on board memory.

The chosen approach proposes a clear separation of concern between the flight and the ground segments. The flight segment has limited knowledge of the system and acts only as executor, while the ground segment has the complete knowledge of the system and is capable of generating sequences of commands to reprogram the flight software, either partition by partition or as a whole.

The adopted approach also focuses more on the functionality and on the mechanisms than on the actual strategy. It seems indeed unrealistic to embrace all possible situations regarding e.g. functional chains or partition boot time criticality within the scope of the present study. On the other hand, the OBSM solution is considered flexible enough to be adapted to most

strategies that can be envisaged, by a combination of simple commands.

## V.4 UNIQUE IMAGE IDENTIFICATION

In order to perform the maintenance of an IMA-SP system, it is necessary to uniquely identify the elements of the system: partitions, hypervisor and configuration. On ground, such information is very important to check, as precondition, whether a patch is applied on the correct/expected image. For instance, address 0x40123456 may refer to a variable X in the partition version 1, but would refer to random data in the same partition version 2. Similarly, the location of code usually differs whenever the partition version changes. So whenever writing in memory, the image version should be checked. Note that this information is needed both for partition images and for XtratuM itself.

The problem is that the identification information cannot be deduced from the non-volatile memory where the partition images to be deployed are located. Indeed, such images are subject themselves to patches, that may or may not have been applied. So, the image in non-volatile memory and the actual, deployed image in RAM can be different (it can only be determined just after a system boot, where all deployed images are the same as the non-volatile images).

Even from the non-volatile information, it is not simple to know what the current image is. Indeed the boot process deploys the image in two parts: first an original image and then possibly a patch applied to the original image. And, there is an infinite number of ways to get the final image 3: original image 1 + patch 1 = final image 3; original image 2 + patch 2 = final image 3; …

The solution would be that each partition (including the hypervisor itself) has an associated ID that could be used by the OBSM tool in order to perform the verification that a patch is applied on the correct image and more generally that what is currently executing on board is what is expected on ground. Such an ID should be made available and observable in the system (potentially all image IDs can be centralized at OBSM level). It should not only be a version number, but should include e.g. a checksum of the image, checksum that should be fixed for a given source code (i.e. no variation due to the compilation time…). This should become a requirement for all IMA-SP partitions and hypervisors.

## V.5 INDEPENDENCE OF THE HARDWARE

A first question that was raised when designing the OBSM was whether this would be the responsibility of a dedicated partition, or if this would be done in the frame of a broader partition in charge of several system aspects (potentially FDIR, basic I/O …). As one of the main goals of the study was to develop a component that could be re-used, it was considered best to have the OBSM partition as a dedicated, stand-alone partition.

In the proposed OBSM solution, there is also no dependency of the OBSM partition towards the underlying hardware/avionics interfaces. The OBSM partition communicates only with a third-partition (the Central DHS in our case) responsible for the TM/TC handling and for accessing the EEPROM. This independence with respect to the hardware is mandatory if the OBSM partition intends to be reused in the frame of other projects, with possibly different avionics architecture and different hardware interfaces.

It must however be noted that, from a system architecture point of view, the OBSM services are so tightly related to the avionics (TM/TC link, access to memory) and to other functional aspects of the system such as FDIR that it would also make sense to have a single partition responsible for both FDIR, maintenance and I/O communications.

As an example, in the frame of this study, the OBSM partition is handling requests to update the EEPROM. A single request is able to upload 2048 bytes into the EEPROM. The actual memory copy is performed by the Central DHS (data is provided back and forth through queuing ports), which requires extra copies and provokes extra delays. Moreover, the actual EEPROM is organised as pages of 64 KB that can only be written as a whole: it means that for each 2048 bytes to be written by the OBSM partition, the system actually writes 64 KB in EEPROM. So, in practice, the design has led to a drastic drop of performance and a raise of response time for any update into EEPROM. In addition, it shall be noted that the lifetime of the FLASH memory should also be taken into account. Indeed, erasing and writing the complete page for only 2KB of data (or less) might cause memory wear-outs during the lifetime of the FLASH memory. It is also a potential limitation that must be included at system level. Indeed, a more sophisticated design may be imagined to improve the management of the non-volatile memory (including buffering or arm and fire system for example).

Such problems are difficult to anticipate in a generic OBSM partition, and they cannot be overcome without breaking the generic aspects. For this reason, the trade-off between separate OBSM partition and OBSM as part of the CDHS should still be evaluated in future projects.

## V.6 UPDATE OF THE GUEST OS AND TSAL

There is a limitation in XtratuM concerning the separation of the Guest OS and TSAL. The limitation is mainly linked to the binary tools of the XtratuM SDK. It

is currently not possible to separate from the partition binary (or the container) the guest OS, XAL or any parts of the partition.

Therefore, the individual update of the Guest OS, XAL or XtratuM is not possible (or very complicated). A new image needs to be rebuilt and the complete partition needs to be reloaded in flight. As a result, it is not possible to modify TSAL and the Guest OS without affecting the application software. This would require that partition software be divided into at least two sub-images, one for the application software and one for the TSAL and Guest OS.

### V.7 MMU ASPECTS

For recall, the target processor for this study is a mono-core LEON-3 processor with MMU. Since the OBSM partition and OBSM Ground Tool both address memory, it is very important to know whether virtual memory or physical memory addresses are used.

The OBSM partition needs an access to the whole memory of the system: it is a pre-requisite to perform the system maintenance.

The memory has been managed as follows by the OBSM partition and OBSM Ground Tool:

- OBSM partition accepts memory read/write/check commands addressing either physical memory or logical (partition) memory;
- OBSM partition knows (via configuration file) the physical address of the non-volatile memory (EEPROM), such that memory write into such areas can be redirected to the Central DHS partition through queuing port;
- OBSM partition can access partition memory using logical addresses via dedicated hypercall;
- OBSM partition can access physical memory by translating the physical address into a logical address of the OBSM partition (in case there is not a direct mapping in the hypervisor configuration), using translation information provided by the hypervisor;
- The OBSM partition accesses the hypervisor itself using physical memory addresses (but not supported by the current version of XtratuM);
- OBSM Ground Tool has access to the hypervisor configuration and knows how to convert between physical and logical address for all partitions;
- Partition images are stored in ELF format by the OBSM Ground Tool: all the addresses referring to the actual partition code, data and entry point are virtual/logical addresses;
- OBSM Ground Tool generates commands using physical memory addresses for direct memory modification of partition configurable parameters;
- OBSM Ground Tool generates commands using physical memory addresses for memory modification of the binary images and headers in non-volatile memory.

In order to ease integration and debugging, the whole project has been executed with physical addresses and logical addresses being equivalent. The complete system has not been tested with different physical and logical addresses; the design should be robust against it.

## VI. CONCLUSION

The objectives of this study have been fully achieved for what concerns the boot approach and the maintenance. The chosen solution is adequate as it allows a wide variety of strategies to be adopted for system boot and system maintenance without any modification of the OBSM partition itself. This flexibility is possible by providing generic, low-level functionality at the OBSM partition level. Higher level functionality remains on the OBSM Ground Tool level, which can be more easily enriched in order to cope with future needs. In particular, the OBSM solution would allow coping with:

- Partitions or systems with critical boot/reboot time;
- Functional chain management;
- Partition or system boot/reboot on multi-core or multi-processor computers;
- Maintenance of system using different hypervisors (not only XtratuM).

The current OBSM solution has however a few limitations:

- Compression/decompression of binary images is foreseen but is not yet supported. The implementation is to be done. The implementation and the testing of such functionality have not been addressed by this study. It is considered out of the scope;
- OBSM partition has no access to XtratuM memory (due to XtratuM design).

In the frame of the OBSW maintenance, modifications have been brought to XtratuM. Few functionality allowing the handling of partition load outside XtratuM, including adequate partition reload hypervisor call.

The OBSM partition and OBSM Ground Tool have been successfully validated and demonstrated. The OBSM solution has been developed according to ECSS-E-ST-40C standard: category C for OBSM partition and category D for OBSM Ground Tool.

From the architecture point of view, having an OBSM partition separated from the central DHS is mandatory if the goal is to reuse it on different platforms. However, from a system point of view, the OBSM services are so tightly related to the avionics (TM/TC link, access to memory) and to other functional aspects of the system

such as FDIR that it would also make sense to have a single partition responsible for both FDIR, maintenance and I/O communications. So, unfortunately, the OBSM partition may not be usable as is on any system: the use of a dedicated, stand-alone OBSM partition should therefore still result from a trade-off in future projects.

The present study has tackled aspects related to functional chains in a partitioned system. It is a broad topic, as functional chains in the partitioned system depend both on the avionics architecture, the partition system architecture and the actual system functionality. The functional chains are also closely related to the FDIR approach. It is a topic that deserves further studies.

In terms of system coherency and observability, each partition should be required to provide a unique identifier (version/checksum) to be identified in an unambiguous way on ground and to ensure that the overall IMA-SP system remains coherent. The system coherency concept has not been pushed to its limits in the present project and several improvement tracks have been proposed.

Aspects related to multi-core and multi-processor computers have not been considered here. Other studies exist or are planned on this topic: it would be interesting to make the link with the OBSM.

The experiment of partitioning PROBA-2 provides good lessons learnt. Despite the fact that new components such as XtratuM environment introduced in the development, the study has been managed quite smoothly in overall. Of course the experiment was very difficult at some points, but at the same time it was very interesting in many aspects.

Finally, the development and the design of the partitions generate a lot of problems but adequate solutions (or sometime workarounds) have been figured out. Small applications have already been partitioned in the past but software with the scale of PROBA-2 was really a live action experiment for the maintenance of critical software of space applications.

## VII. REFERENCES


[1] Rushby, J., Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance, SRI International Technical Report, 1999

[2] ARINC 653-1 Avionics Application Software Standard Interface, http://www.arinc.com, 2005

[3] PROBA-2, http://sci.esa.int/proba2/