Partitioned Embedded Architecture based on Hypervisor: the XtratuM approach

S. Peiró, A. Crespo, I Ripoll, M. Masmano
Instituto de Automatica e Informatica Industrial
Universidad Politecnica de Valencia
Valencia, Spain
{*speiro, acrespo,iripoll,mmasmano*}*@ai2.upv.es*

Abstract—**Partitioned sofware architectures were conceived to fulfill security and avionics requirements where predictability is extremely important. Both, the availability of new processors and an increased necessity of security, have opened new possibilities to use efficiently this approach. Avionic industry has consolidated the Integrated Modular Avionics (IMA) as a solution to manage the software growth in functionality and in efficiency. Now, the aerospace sector is adapting these concepts on its developments. One of the solutions used to achieve partitioned systems is based on virtualisation techniques. In this paper we present XtratuM, a bare-metal hypervisor which implements para-virtualization and dedicated device techniques. XtratuM provides a virtual machine that is 'near' the native one. It permits to execute a set of partitions, containing each one an operating systems and its applications. Security is based on the temporal and spatial isolation properties provided by the hypervisor. This paper describes the main design criteria used to achieve temporal and spatial partition isolation and an approach to extend the trusted environment from the hardware level to the hypervisor level in order to verify the temporal and spatial isolation properties**

Keywords-**hypervisor; real-time; secure kernel;**

I. Introduction

The availability of new processors for embedded applications has raised new possibilities for these applications. Currently, the embedded applications have more features and, as a consequence, more complexity. There exist a growing interest in enabling multiple applications to share a single processor and memory. To facilitate such a model the execution time and memory space of each application must be protected from other applications in the system.

Partitioned software architectures have evolved to fulfill security and avionics requirements where predictability is extremely important. The separation kernel proposed in [1] established a combination of hardware and software to allow multiple functions to be performed on a common set of physical resources without interference. The MILS (Multiple Independent Levels of Security and Safety) initiative is a joint research effort between academia, industry, and government to develop and implement a high-assurance, real-time architecture for embedded systems. The technical foundation adopted for the so-called MILS architecture is a separation kernel. Also, the ARINC-653 [2] standard uses these principles to define a baseline operating environment for application software used within Integrated Modular Avionics (IMA), based on a partitioned architecture.

Virtual machine technology [3] is a secure and efficient way to build partitioned systems. A virtual machine (VM) is a software implementation of a machine (computer) that executes programs like a real machine. **Hypervisor** (also known as virtual machine monitor VMM [4]) is a small software layer (or a combination of software/hardware) that enables to run several independent execution environments or partitions in a single computer. The key difference between hypervisor technology and other kind of virtualisation (such as java virtual machine or software emulation) is the performance. In bare-machine hypervisors the overhead can be very low maintaining the throughput of the virtual machines very close to the native hardware.

The low overhead and the reduced size of the hypervisor can be considered as an appropriated solution to achieve secure systems if it is designed following strict design criteria to meet security requirements. Its correctness can be sufficient to ensure the security of the system as a whole or, at least, the security of a set of trusted partitions. In a partitioned system, the partitions can accommodate different kinds of applications: real-time, trusted, non trusted, etc. As consequence, the partition's operating system can be tailored to provide a set of specific services to its applications.

In this paper we present a solution for partitioned based on a bare-metal hypervisor called XtratuM. It has been designed specifically for critical real-time systems following a set of requirements for secure space applications based on the ARINC-653 standard. In the next section, we present a review of the virtualisation techniques for embedded real time systems. Section III presents the main design criteria. Also, we analyse the processor dependencies and and the virtualise services to the partitions. Section III-A describes the hypervisor architecture and the services provided to the partitions. Finally, we build a model based on finite state machines that permits to analyse and validate the temporal and spatial properties. Finally some conclusions are enumerated.

II. Real-time embedded partitioned systems

In the best of the author's knowledge, the separation kernel approach was the first technique used to achieve highly secure systems. It is a natural extension for the already existing operating systems. A partitioned kernel is basically a real-time operating system with some modifications to enforce stronger isolation between processes.

Platform virtualisation is an enabling technology that can be used (among several other uses) to cover some security aspects of the application. Depending on the type and requirements of the application, some virtualisation techniques work better than others. Although ideally a hypervisor which provides full and transparent virtualisation is the most desirable one. In the case of embedded systems with real-time constraints, the partial virtualisation (para-virtualisation method) seems to be the most appropriated choice.

One important aspect is the amount of critical code. In the case of executing the hypervisor on top of an OS, the hosted OS is also in the path of the critical system, and therefore, it shall also be certified. Additionally, it is important to consider the peripheral sharing policy. In some cases, when the policy for sharing a peripheral is user-specific (or when the user requires a fine grain control over the peripheral), it is better to allocate the native peripheral to a specific partition. This technique (named *dedicated devices*) is widely used in embedded systems when a device is not shared among several partitions, or when the complexity of the driver is that high that it does not worth including it in the hypervisor layer. When the peripheral is shared by several partitions, the system can be also organised by using a specific partition (IOserver) that access directly to the shared peripherals and provide services to other partitions.

An alternative to the hypervisor approach is the μkernels. This architecture has two main components: the ”separation kernel” and the ”system software”. These services can be used to build several different operating systems, resulting in a virtualized system. The μkernel approach started with the Mach μkernel [5]. One of the most representative implementation of a μkernel is the L4 [6]. One of the main drawbacks of this solution is that all application threads are handled by the μkernel introducing, thus a large number of context switches at the μkernel level which has a strong impact on the system performance.

III. XtratuM Overview

XtratuM [7] has been designed to achieve temporal and spatial requirements of safety critical systems. Ported over the LEON2 [8] processor to implement the TSP-based (Time and Space Partitioning) solution for generic payload on-board software proposed by CNES (Centre National d’tudes Spatiales) [9]. TSP-based architecture has been identified as the best solution to ease and secure reuse, enabling a strong decoupling of the generic features to be developed, validated and maintained in mission specific data processing [10].

LEON2 processor is a 32-bit processor core based on the SPARC V8 architecture, suitable for system-on-a-chip (SOC) designs, which can be synthesized in a FPGA. It is used by the European Space Agency and has successfully been used in various commercial and research endeavours.

A. XtratuM Architecture

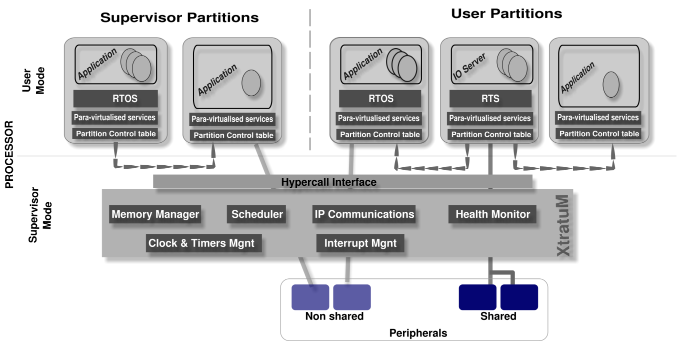

Figure 1. XtratuM architecture.

XtratuM is in charge of virtualisation services to partitions. It is executed in supervisor processor mode and virtualises the cpu, memory, interrupts and some specific peripherals. The figure 1 shows the complete system architecture. The internal XtratuM architecture includes: memory management, scheduling (fixed cyclic scheduling), interrupt management, clock and timers management, partition communication management (ARINC-653 communication model), health monitoring and tracing facilities. Three layers can be identified:

- Hardware-dependent layer: It implements the set of drivers required to manage the strictly necessary hardware: processor, interrupts, hardware clocks, hardware timers, paging, etc. This layer is isolated from the rest through the Hardware Abstraction Layer (HAL). Thus, the HAL hides the complexity of the underlying hardware by offering a high-level abstraction.
- Internal-service layer: These services are not available to the partitions. This layer includes a minimal C library which provides the strictly required set of standard C functions (e.g. strcpy, memcpy, sprintf) and a bundle of data structures. The system boot is also part of the internal services.
- Virtualization-service layer: It provides the services required to support the para-virtualisation services, which are provided via the hypercall mechanism to partitions. Some of these services are also used from other XtratuM modules.

B. Partitions

A partition is an execution environment managed by the hypervisor which uses the virtualised services. Each partition consists of one or more concurrent processes (implemented by the operating system within each partition), sharing access to processor resources based upon the requirements of the application.

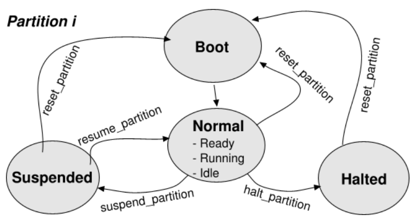

Figure 2. Partition states.

Partitions need to be *virtualised*, that is, changed to collaborate with the hypervisor, in order to be executed on top of XtratuM. For instance, a partition cannot manage directly the state of the hardware interrupts: a try of enabling/disabling directly them raises a trap. Therefore, the partition has to be modified to invoke the adequate hypercalls.

XtratuM defines two types of partitions: *normal* and *system*. System partitions are allowed to manage and monitor the state of the system and other partitions. Some hypercalls cannot be called by a normal partition or have restricted functionality. Figure 2 shows the set of partition states with the events generated by the hypervisor as result of a system partition hypercalls.

Note that system rights are related to the capability to manage the system, and not to the capability to access directly to the native hardware or to break the isolation: a system partition is scheduled as a normal partition; and it can only use the resources allocated to it in the system configuration specification. Table I summarises hypercalls grouped depending on their mission.

Group	Some hypercalls	Super
Partition management	start, stop, reset, resume, shutdown, status	Yes
Health monitor log	open, read, status	Yes
Trace management	open, read, seek, status	Yes
Interrupt management	enable, disable, mask, unmask	No
Clock management	set_timer, get_clock	No
Interpartition communication	create_port, read, write, get_status	No
IO management	input output	No
SparcV8 dependent	atomic_op, flush_regwin, flags	No

Table I
LIST OF GROUPS OF HYPERCALLS.

C. Processor dependencies

In a bare-metal hypervisor, the hardware dependency is an important issue. The presence or absence of some processor features (MMU, or MPU (Memory Protection Unit), supervisor modes, etc.) has a strong impact on its design and posterior implementation.

Some relevant features of the processor have a major impact on the virtualisation. One of these features is the absence of MMU. The LEON2 processor presents this *feature*, it does not includes a MMU but a rather simple memory protection mechanism consisting in two write protection registers (dubbed WPR). These registers enable to define read-only memory areas, raising an exception any time a write operation within these areas is carried out. In addition, this processor implements a IO protection mechanism: it is possible to disable the access to the I/O memory-mapped registers through a bit (IOP bit) in a configuration register.

IV. Trustability enforcement

The hypervisor is a promising technology to build trusted systems. In order to design a hypervisor for safety critical systems, the following design criteria have to be followed:

- Strong spatial isolation: the hypervisor is executed in privilege (supervisor) processor mode whereas partitions are executed in user one. Partitions are allocated in independent physical memory addresses. A partition only can access to its memory areas.
- Strong temporal isolation: It enforces the temporal isolation by using a fixed cyclic scheduler.
- System partitions: some partitions can use *special* services provided by the hypervisor.
- Robust communication mechanisms: partitions are able to communicate with other partitions by using specific services provided by the hypervisor. The basic mechanism provided to the partitions is the port-based communication. The hypervisor implements the link (channel) between two ports or more ports. Two types of ports are provided: sampling a queuing as defined in the ARINC-653 standard [2].
- Interrupt Model: the hypervisor provides a secure interrupt model to the partitions. A partitions cannot interact with native traps. All the interrupts are, in first place, handled by the hypervisor, who is in charge of propagate them to partitions according to the system configuration file.
- Fault management model: faults are detected and handled by the hypervisor.A fault can be defined as the occurrence of a system trap or an event triggered by the hypervisor itself.
- Non-preemtable: in order to reduce the design complexity and increase the reliability of the implementation, the hypervisor is designed as a monolithic, non-preemtable kernel. This decision prevents the occurrence of internal race conditions and facilitates the formal model.
- Resource allocation: fine grain hardware resource allocation is specified in the system configuration file. This configuration permits to assign system resources (memory, I/O registers, devices, etc.) to the partitions.
- Small: The validation and formal verification complexity increases with the number of lines of code. The hypervisor code shall provide the minimum services in order to be as minimal as possible.

- Deterministic hypercalls: All services (hypercalls) shall be deterministic and fast.

A. Interrupt Model

Different manufacturers use terms such as exceptions, faults, aborts, traps, and interrupts to describe the processor mechanisms to receive a signal that requires attention. We will use the term *interrupt* to globally refer to these mechanisms.

In a partitioned system, the hypervisor handles the interrupts (*native interrupts*) and generates the appropriated *virtual interrupt* to the partitions that can be: i) *virtual traps* (as consequence of *traps*); ii) *virtual exceptions* (as consequence of a *exceptions*); and iii) *virtual hardware interrupts* (as consequence of peripheral interrupts).

Not all the *exceptions* are propagated to the partitions. For instance, a memory access error that is generated as consequence of a space isolation violation is handled by the hypervisor which can perform a halt partition action or can generate another different virtual exception (such as memory isolation fault). On the other hand, a numeric error is propagated directly to the partition. *Virtual exceptions* are a superset of the *exceptions* which includes additional exceptions generated by the hypervisor (virtual processor). Some of them are: memory isolation error, IO isolation error and temporal isolation error. Only *virtual hardware interrupts* can be enabled or disabled by partitions.

One of the strategies used to prevent partitions to jeopardise temporal isolation is that partitions cannot access to the interrupt table and cannot interact with native interrupts. All interrupts are directly handled by XtratuM and, when required, propagated to the partitions which define their own *virtual interrupt table*. A second strategy is related to the scheduling, when a partition is scheduled, the *harware interrupts* associated to other partitions are disabled: during the partition scheduling, the hypervisor detects the pending interrupts for the next partition to be executed and raises (emulates) them depending on the partition interrupt mask.

B. Fault Management Model

The Health Monitor (HM) is the part of XtratuM that detects and reacts to anomalous events or states. The purpose of the HM is to discover the errors at an early stage and to try to solve or confine the faulting subsystem in order to avoid or reduce the possible consequences.

HM is invoked as result of a HM_event occurrence. Following scenarios could raise a HM_event:

- An exception has been triggered by the CPU. The exception handler generates the associated HM_event.
- A native interrupt has been received and the temporal or spatial properties are not valid.
- A trap has been received and the temporal or spatial properties are not valid.
- A partition detects an abnormal internal situation and raises a HM_event. For instance, the operating system inside of a partition detects that the application is corrupted.
- When the partition request a hypervisor service (hypercall), the spatial or temporal properties are verified as pre- and post-conditions. If these validations fail, a HM_event is generated.

The HM_event occurrence is the manifestation of an error. XtratuM reacts to the error providing a simple set of predefined actions to be done when it is detected. Once a HM event is raised, XtratuM performs an action that is specified in the configuration file. These actions can be, depending on the causing fault (partition or hypervisor), stop, shutdown or reset a partition, propagate or ignore the event to the partition, halt, shutdown, reset the system. HM events and actions are logged to permit remote attestation.

C. System specification

Deploying a partitioned system presents many challenges related to the system specification, configuration, resource allocation and validation. This configuration process involves two different type of roles: the system integrator and the partition developers. The integrator is responsible of system definition and resource allocation. In XtratuM, the system specification is detailed in a XML file that specifies:

- XMHypervisor: species the board resources and the hypervisor health monitoring table. It includes: the list of memory regions allocated to XtratuM, the processor frequency and the scheduling plan.
- PartitionTable: this is a container element which holds all the partition elements. A partition element specifies the resources which belongs to the partition: the memory areas, the HW-interrupts, the IO ports, a list of communication ports and the temporal constraints.
- Channels: a list of channels which defines the interconnections between the partitions' ports. For each channel, the following information is specified: a channel identifier, the type, the input and output ports, the maximum message size, the maximum number of messages (queuing channels).
- Devices: contains the conguration of the virtualised devices.

The XML file is parsed and validated against the main system properties of a partitioned system. The result of this validation process is a set of automatically generated data structures (XM_CT) that will be compiled in the deployment phase, jointly with the binaries of the partitions and the hypervisor itself, to generate the system container to be deployed to the target.

V. Hypervisor model

An initial assumption is that the underling hardware is trustworthy. It means that the internal processor registers work properly if they are used in a correct way. For the temporal and spatial isolation purposes, we assume that: i) the access to the processor registers is only allowed when the processor is in privileged mode. The processor mode can be set or unset by accessing the control processor status (PMS); ii) the processor memory protection registers (WPR) raises an exception when an instruction tries to write in a protected memory area; iii) a specific timer is exclusively used by XtratuM to control the slot duration; iv) the IOP bit in the control processor status permits to enable/disable the access to the IO ports; and v) the interrupt vector is handled exclusively by XtratuM, therefore any access/modification can be done only when the processor is in privileged mode.

XtratuM can be defined as a finite state machine [11] given by $(\Omega, \alpha, S, S_0, \theta, F)$ where:

- Ω is the system configuration. It is automatically generated from the system specification (XM_CT).
- α is the input alphabet described by the set of events which are accepted by the hypervisor (hardware interrupts, exceptions and traps).
- S is a finite non-empty set of states. These finite states are result of the scheduling plan. Each state corresponds to the execution of a partition in the scheduling plan and has associated a relative initial and final time with respect to the MAF origin. From the temporal and spatial isolation properties only three events are significant: *next_slot*, an exception that can perform a system halt (HM action), and a trap (hypercall).
- S_0 is the initial state which correspond to the hypervisor state after booting and loading the partitions.
- θ is the state-transition function given by $\delta : S \times \Sigma \rightarrow S$. A function $f^{\Omega}(S_i)$ extracts the set of hardware parameters associated to the S_i state.
- F is the final state that corresponds to the system halt.

Figure 3 shows the set of states of the hypervisor generated from the static scheduling plan defined in the specification. Each state S_i models the status of the hypervisor when a partition P_i is under execution. The transition from one state to the next one, is consequence of the *next_slot* occurrence which is the slot duration defined in the scheduling plan. This event is the hardware interrupt associated to the slot duration timer. After the arrival of this event (interrupt), the hypervisor stores the context of the previous partition, selects the next partition to be executed, extracts from the XM_CT table the harware context to guarantee the spatial and temporal isolation, and then the control is transfered to this partition if the partition status is in ready state.

During the execution of a partition, the hypervisor can be invoked as consequence of an interrupt, exception or hypercall (trap). However, the hypervisor invocation has no effect from the temporal and spatial isolation point of view, mainly because the hypervisor state remains in the same state until the arrival of the next_slot event. Anyway, a pre- and post-

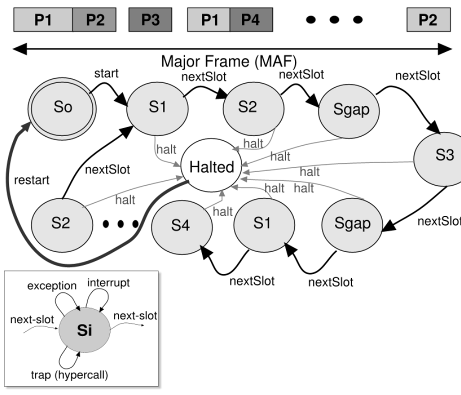

Figure 3. Finite set of states.

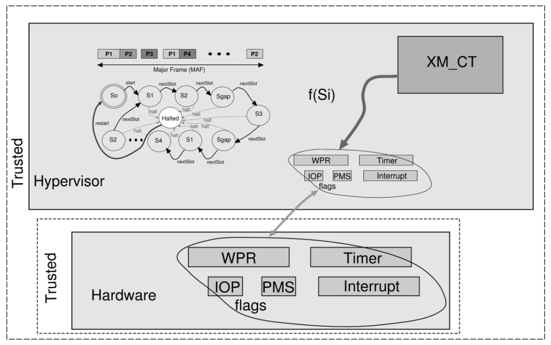

Figure 4. State variables.

condition validation of the temporal and spatial properties is performed each time the hypervisor is invoked. A violation of these properties raises the appropriated HM_event.

Figure 4 depicts a view of the hypervisor model and the state variable used to extend the trustworthiness.

A. Spatial isolation properties

The basic concern of spatial isolation is to detect and avoid the possibility that a partition could access to another partition's memory area for reading or writing. Hardware provides some basic mechanisms to guard against violations of spatial isolation. Spatial isolation properties are conditions that permit to guarantee that the hardware mechanisms are set to the appropriated values when a partition is under execution. These conditions have to compare the memory area region of the current partition against the processor memory protection registers in the LEON2 architecture.

The spatial isolation property states that the data processing occurred in any partition i cannot access to any memory address outside of the its address memory region. This is granted if the hardware mechanisms are used according to the principles announced above.

Property 1: Supposing that the hypervisor is in state S_i and the next state will be S_j as consequence of the event $next\_slot \in \alpha$ occurrence. In this situation, at the entry of *the hypervisor: $PRegs = f^{\Omega}(S_i)$, and at the output of the hypervisor $PRegs \leftarrow f^{\Omega}(S_j)$).*

were $f^{\Omega}(S_i)$ the function that extracts the $PRegs$ values from a XM_CT and $PRegs$ are the hardware registers. For $\forall e \neq next\_slot \in \alpha$ the state is not changed.

Also, the spatial isolation refers to the IO memory access. In the LEON2 processor, the IO memory access protection is a global mechanism. To guarantee the isolation, a direct IO memory access is forbidden to partitions. It is only provided through a specific set of hypercalls which supervises the correct access by part of partition the IO memory mapped addresses. The hypervisor has to validate that the hardware mechanisms are enabled.

Property 2: Independently of the initial and final states, when an event $e \in \alpha$ occurs, the IO memory protection ($IOP \in PRegs$) bit is set to 1 at the entry of the hypervisor $IOP = 1$ and again $IOP = 1$ when the control is transferred to the partition. During the execution of the hypervisor $IOP = 0$.

B. Temporal isolation properties

Temporal isolation refers to the system ability to execute several executable partitions guaranteeing that the execution of each partition does not depend on the temporal behaviour of any other partitions. Temporal isolation enforcement is achieved designing a schedulable plan and guaranteeing that it is executed as specified. The hypervisor scheduling is responsible of the correct execution of the plan. XtratuM implements a static (cyclic) scheduling following the ARINC 653 specification [2] which defines a cyclic scheduling for the global scheduler and a preemptive fixed priority-based policy for the local scheduler (at partition level).

The concern of temporal isolation is to guarantee that a partition is executed only in the intervals specified in the scheduling plan. Moreover, interrupts allocated to other partitions shall not impact on the partition execution. If the execution plan is guaranteed, there is not possibility for any partition to monopolise the CPU or crash the system. Other scenarios that could cause a partition to fail to relinquish the CPU on time include simple schedule overruns. Slot duration is controlled by a timer that is used exclusively by the hypervisor and cannot be influenced by the partitions timers which are attached to a second hardware timer.

The temporal isolation properties can be defined as:

Property 3: At any instant of the state S_i, the clock value is in the interval specified by the slot interval. $current\_clock \in f^{\Omega}_{clk}(S_i)$

where $f^{\Omega}_{clk}(S_i)$ is the function that returns the interval of the slot associated to S_i).

VI. Conclusion

Complexity of embedded systems within satellites is growing dramatically. Payload software in that context has evolved during the past ten years from simple data processing, mainly formatting and transferring to ground, into complex data processing and autonomous treatments. In this context, TSP (Time and Space Partitioning) based architecture has been identified as the best solution to ease and secure reuse, enabling a strong decoupling of the generic features to be developed, validated and maintained, XtratuM is used as enabling technology for TSP developments.

In this paper we have presented a hypervisor specifically designed for safety critical applications. XtratuM has been designed following strict criteria to guarantee the temporal and spatial isolation properties as defined in the ARINC 653 standard and the MILS approach. XtratuM defines a virtual machine very close to the native where the main resources are virtualised. The executable entities (partitions) are executed on top of a virtual machine. The virtual machine defines an interrupt model to the partitions which is a superset of the system interrupts. The hypervisor defines its own virtual interrupts which are delivered to the partitions. Also, there is a Fault Management model which is directly related to the health monitor included in the internal architecture. This health monitor is in charge of the fault detection and error isolation through a set of action that are directly related to the partition under execution. Finally, it has been presented an extension of the trusted environment from the hardware platform to the hypervisor limits. The presented model is based on the hardware mechanisms provided by the specific processor (LEON2) and an exclusive use of these mechanisms by the hypervisor. This model is based on a finite state machine as formalism.

References

[1] J. Rushby, "Design and verification of secure systems," vol. 15, no. 5, Pacific Grove, California, Dec 1981, pp. 12–21.

[2] *Avionics Application Software Standard Interface (ARINC-653)*, March 1996, Airlines Electronic Eng. Committee.

[3] I. Corporation, "IBM systems virtualization. Version 2 Release 1 (2005)," http://publib.boulder.ibm.com/infocenter/-eserver/v1r2/topic/eicay/eicay.pdf.

[4] R. Goldberg, "Survey of virtual machine research." *IEEE Computer Magazine*, vol. 7, no. 6, pp. 34–45, 1974.

[5] D. B. Golub, R. W. Dean, A. Forin, and R. F. Rashid, "Unix as an application program," in *USENIX Summer*, 1990, pp. 87–95.

[6] J. Liedtke, "On microkernel construction," in *Proceedings of the 15th ACM Symposium on Operating System Principles (SOSP-15)*, Copper Mountain Resort, CO, Dec. 1995. [Online]. Available: http://l4ka.org/publications/

[7] M. Masmano, I. Ripoll, and A. Crespo, "Introduction to XtratuM," 2005. [Online]. Available: http://www.xtratum.org/doc/papert/xtratum_intro.pdf

[8] G. Research, "Leon2 processor users manual," http://www.gaisler.com.

[9] P. Arberet, J.-J. Metge, O. Gras, and A. Crespo, "TSP-based generic payload on-board software," in *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[10] P. Arberet and J. Miro, "IMA for space : status and considerations," in *ERTS 2008. Embedded Real-Time Software.*, Jannuary. Toulouse. France 2008.

[11] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.