



Distributed Real-time Architecture for Mixed Criticality Systems

XtratuM support of enhanced hypervisor layer services: description and interfaces **D 2.3.1**

Project Acronym	DREAMS	Grant Agreement Number	FP7-ICT-2013.3.4-610640		
Document Version	1.0	Date	31.03.2015	Deliverable No.	2.3.1
Contact Person	Javier O. Coronel	Organisation	FENTISS		
Phone	+34 963294704	E-Mail	jcoronel@fentiss.com		

Contributors

Name	Partner
Javier Coronel	FENTISS
Alfons Crespo	UPV
Miguel Masmano	FENTISS
Vicent Brocal	FENTISS

Table of Contents

Contributors	2
Abstract	6
Terms, definitions and abbreviated Terms	7
1 Introduction.....	9
1.1 Structure of the deliverable	9
1.2 Relationship to other DREAMS Deliverables.....	9
2 Virtualization Overview	10
2.1 Virtualization layer	11
2.1.1 Full virtualization	12
2.1.2 Para-virtualization	13
2.1.3 Hybrid Virtualization.....	13
2.2 I/O Virtualization	13
2.2.1 IOMMU Virtualization	13
2.2.2 Device and I/O Virtualization	14
2.3 Guest Operating System.....	14
3 XtratuM hypervisor overview.....	16
3.1 Basic properties.....	17
4 XtratuM – Software design overview	19
4.1 Software static architecture	19
4.1.1 System deployment.....	19
4.1.2 System components	20
4.1.3 System states.....	23
4.2 Partition overview	23
4.2.1 Partition operation	24
4.2.2 Types of partitions.....	24
4.2.3 Names and identifiers	24
4.3 Partition scheduling.....	25
4.3.1 Multiple scheduling plans.....	25
4.4 Inter-Partition communications (IPC).....	26
4.5 Health Monitor (HM).....	27
4.6 Inter-Partition Virtual Interrupts (IPVI)	28
4.7 Interfaces context required by XtratuM	29
4.8 Interfaces context provided by XtratuM	29
4.8.1 Hypercalls	29
4.8.2 Binary interfaces.....	31
4.8.4 Partition control table (PCT).....	32
4.8.5 Virtual Interrupts	33

4.8.6	Fault management model	36
4.8.7	Partition image header	37
5	Booting process	39
5.1	Hypervisor boot	39
5.2	Partition boot	40
6	System configuration	41
6.1	XtratuM subjects, objects and privileges	41
6.1.1	Subject identification	41
6.1.2	Exported resource identification	42
6.1.3	Exported resource access mechanism	43
6.1.4	Operations on exported resources	43
6.1.5	Partitions and the Partitioned Information Flow Policy (PIFP)	44
6.1.6	Access matrices	44
6.1.7	Subject temporal allocation	47
6.1.8	Subject memory areas allocation	47
6.1.9	Subjects and virtualized exported resources	47
6.1.10	IPC exported resources	48
6.1.11	Devices exported resources	48
6.2	Configuration file specification	48
6.2.1	Element HwDescription	51
6.2.2	Element XMHypervisor	57
6.2.3	Element ResidentSw	59
6.2.4	Element PartitionTable	61
6.2.5	Element Channels	64
6.2.6	Basic types	65
7	Secure State and Secure Operations	73
7.1	Secure State	73
7.2	Insecure state	74
7.3	Trustability enforcement	74
7.4	Test for secure states	75
7.4.1	Abstract machine test (AMT)	75
7.4.2	Basic platform tests	75
7.4.3	Maintenance tests	76
8	DREAMS Abstraction Layer (DRAL)	77
8.1	DRAL	78
8.1.1	System Management Services	78
8.1.2	Partition Management Services	78
8.1.3	Process Management	79
8.1.4	Time Management Services	79

8.1.5	Inter-Partition Communication Services	79
8.1.6	Intra-Partition Communication	80
8.1.7	Scheduling Services	80
8.1.8	Monitoring Services (Health Monitor)	80
8.1.9	Configuration services	80
9	Bibliography.....	82
APPENDIX 1	83

Abstract

This document describes the internal design of the XtratuM hypervisor. This presents an overview of the main features, services and properties of the virtualization layer focused to the development of mixed-criticality applications in distributed multicore platforms.

In this document the *DREAMS Abstraction Layer* specification is included, where the definition of the main features and services of this software layer are presented. This layer is intended to deliver DREAMS services to the application regardless of the software layer below.

Terms, definitions and abbreviated Terms

Definition of terms.

Application	A set of cooperating tasks which together perform a coherent function, e.g. an avionics function. The scope of an application, i.e. which software functions it is made of, is defined by the system architectural design activity.
Computing platform	Environment in which applications execute; it provides Computing Resources to every application. An application has no other interface than the one provided by the computing platform.
Computing resource	Resources are the totality of all hardware, firmware and software and data that are executed, utilized, created, protected or exported by the Computing Platform
Configuration file	The file that describes the system configuration in a user-friendly format (XML)
Configuration vector	The binary internal representation of the configuration file
Error	An error is the part of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service.
Failure	A failure is an event that occurs when the delivered service deviates from correct service.
Fault	A fault is the adjudged or hypothesized cause of an error.
Hypercall	The service (system call) provided by the hypervisor. The services provided are known as para-virtual services.
Hypervisor	The layer of software that, using the native hardware resources, provides one or more virtual machines (partitions).
Partition	Also known as "virtual machine" or "domain". It refers to the environment created by the hypervisor to execute user code.
Partition code	Also known as "guest". It is the code executed inside a partition. Usually, the code is composed of an operating system and a set of processes or threads. Since application code relies on the services provided by the OS, we will assume that the partition code is an operating system (or a real-time operating system).
Resident software	The booting software that is executed directly in ROM memory right after a system reboot, also referred as boot-loader or firmware. Among other tasks, it is in charge of loading RAM memory and the initial partitions.
Slot	See temporal window.
System partition	A partition that has extra capabilities to manage and control the system, and other partitions. Originally these partitions were named "supervisor partitions" but to avoid confusion with the processor modes it was renamed as "system partitions".
Temporal window	Time interval in which a partition is scheduled. A temporal windows is specified with an initial time point and an interval. Also known as slot.
Tile	A tile can be processor cluster with several processor cores, caches, local memories and I/O resources. Alternatively, a tile can also be a single processor core or an IP core (e.g., memory controller that is accessible using the NoC and shared by several other tiles).

Acronyms and Abbreviations

ABI	Application Binary Interface
API	Application Programming Interface
ARINC	Aeronautical Radio, Incorporated
CPU	Central Processing Unit
DRAL	DREAMS Abstraction Layer
guestOS	Guest Operating System
HM	Health Monitor
IRQ	Interrupt Request
libXM	XtratuM library
OS	Operating System
PCT	Partition Control Table
PIFP	Partitioned Information Flow Policy
SKPP	Separation Kernel Protection Profile
TSP	Temporal and Space Partitioning
UART	Universal Asynchronous Receiver-Transmitter
UML	Unified Modeling Language
VLayer	Virtualization Layer
VM	Virtual Machine
XM_CF	XtratuM Configuration File
XSD	Xtratum Security Functions
VCPU	Virtual CPU
VFPU	Virtual Floating Point Units

1 Introduction

This document presents an overview of the XtratuM hypervisor design to achieve the basic properties of real-time applications based on multicore systems. The principles of the virtualization layer design presented here are general enough to address application domains in order to build a solution that allows the engineering of mixed-criticality applications over the envisioned distributed multicore platforms.

This document sketches a summary of the architectural design of the virtualization layer to be developed in the context of the DREAMS project. A detailed XtratuM design can be gathered in internal reports and technical manuals of FENTISS, some of them released into the project. This deliverable is mainly focused in the XtratuM design for the DREAMS harmonized platform using the Zynq-7000 board. However, the basic properties and principles of this design can be used as reference for other platforms using XtratuM. Other development systems into the DREAMS project that will also use XtratuM are the PPC T4240-QDS and the Industrial PC Automation PC910, which include hardware-assisted virtualization. Specific features of XtratuM for those architectures will be presented in subsequent DREAMS deliverables such as *D2.3.4 Hypervisor adaptation and drivers for local resource manager*.

Additionally, in this document the DRAL (*DREAMS Abstraction Layer*) specification is also included. A definition of the features and services provided by this software layer is presented. This layer is intended to deliver DREAMS services to the application in a transparent way of the virtualization layer.

1.1 Structure of the deliverable

This document is organised as follows: section 2 offers an outline of the world of the virtualization. Section 3 presents a XtratuM hypervisor overview where the hypervisor is introduced and the basic properties to be achieved by XtratuM are pointed out. Section 4 outlines the software design overview of the virtualization layer. It describes the software static architecture and the main features of the hypervisor such as a partition overview, scheduling, health monitor and communication schemas, it also details the interface to the hardware and partitions. Section 5 details the booting process of the virtualization layer and partitions. Section 6 describes the scheme of the configuration file. Section 7 presents the approach to formalize a security model. Finally, section 8 outlines the DREAMS Abstraction Layer.

This document is complemented with an Annex that provides specification of the DREAMS Abstraction Layer. As this annex contains a high number of pages, we considered that it is more convenient to provide it as separate document: *D2.3.1 Annex - DREAMS Abstraction Layer (DRAL) Specification* (Annex 1).

1.2 Relationship to other DREAMS Deliverables

This document is an important input for the development of the demonstrators in WP6 and WP7. Additionally, it serves as input for *D1.5.1 Intermediate integration of DREAMS platform with virtual platform prototype* and *D2.3.4 Hypervisor adaptation and drivers for local resource manager*. Furthermore, this deliverable is an input for the WP5, where the XtratuM design is used as modular safety case of a hypervisor. DRAL specification uses as main input the deliverable *D1.2.1 Architectural Style of DREAMS*.

2 Virtualization Overview

A quick overview through the world of the virtualization is presented in this section. This section is developed mainly from the experiences around the XtratuM hypervisor.

Although the virtualization technology has been used in mainframe systems since 60's; the advances in the processing power of the desktop processors in the middle of the 90's opened its adoption in the PC market. The embedded market is now ready to take advantage of this promising technology.

Virtualization is a generic term that refers to the abstraction of the computer resources. The current state of the virtualization technology is the result of a convergence of several technologies: operating system design, compilers, interpreters, hardware support, etc. As a result, there are several competing/complementing technologies that can be used to build a virtual execution environment (or virtual machine, VM).

Three main alternative virtualization techniques can be highlighted:

- Full-virtualization.
- Para-virtualization.
- Hybrid Virtualization.

The implementation of these techniques will depend mainly on the support of the hardware architecture. The architectures can be:

- Non-Virtualizable
- Virtualizable
- Virtualizable using hardware virtualization extensions

Virtualizable architectures are those one that can be purely virtualized with trap-and-emulate model, that is, every privilege operation generates a trap that can be handled by the hypervisor, emulating its correct operation. However, non-virtualizable architectures can still be virtualized by using complex software techniques such as binary translation (BT).

Additionally, hardware virtualization extensions have been introduced on some architectures in order to include virtualization support or to improve the performance and simplify virtualization techniques. The latter is commonly called *hardware-assisted virtualization*.

Full-virtualization simulates one or more instances of an underlying hardware environment. From the point of view of "guests" OS, it is executed as on a native hardware.

Para-virtualization incorporates direct communications between the "guest" OS and the hypervisor. This virtualization technique does not simulate hardware but involves modifying the "guest" OS to replace non-virtualizable instructions with hypercalls that communicate directly with the virtualization layer hypervisor.

Hybrid Virtualization combines both the *full-virtualization* and *para-virtualization* techniques in order to take advantage of each technique.

In the development of partitioning systems three main groups should be taken into account: the user application, the guest operating system, and the hypervisor/VMM. Depending on the virtualization techniques used in the virtualization layer and the hardware virtualization extensions available on the development platform, the groups listed above will have a different impact from the point of view of performance and efficiency, compatibility and portability, complexity and maintainability. These aspects will be discussed in the following subsections. Note that although some implementations could include only a bare application and a hypervisor/VMM, this section addresses the most generic partitioned systems structure.

2.1 Virtualization layer

There are two main environments to run the virtualization layer: hosted or bare-metal architectures. A hosted architecture installs and runs the virtualization layer on top of an O.S. and it is executed as an additional application on the system. Bare-metal architecture, also known as hypervisor, installs the virtualization layer directly on the hardware. A bare-metal architecture is more efficient than a hosted architecture and delivers greater scalability, robustness and performance. This is because the hypervisor has direct access to the hardware resources rather than going through an operating system, where the efficiency could depend on the type of O.S. used as host.

This subsection describes the benefits and drawbacks of use the different alternative techniques of virtualization in the development of the virtualization layer:

- Full virtualization.
- Para-virtualization.
- Hybrid Virtualization.

Table 1 - Comparison in the development of virtualization layer

	Full Virtualization	Hybrid Virtualization	Para-virtualization
Method	Hw-assisted: Exit to Root Mode on privileged instruction. Non-Hw assisted: Privileged Instructions, and/or binary translations. Use virtual devices.	Hybrid: hypercalls combined with methods used on full virtualization.	Hypercalls: service requests to the hypervisor. The hypercalls is the equivalent to system calls in O.S.
Complexity	High: Developing of firmware is required. Medium: using hardware assisted virtualization	Medium: Developing of firmware and hypercalls	Low: Developing of hypercalls
Performance	Poor with first generations of x86 virtualization extensions. Good with the latest generation virtualization extensions for X86. Poor/good using binary translation or trap-and-emulate technique.	Good. Although it depends on the generation hardware virtualization technology. This technique could increase the determinism in the execution.	Better in some cases
Portability and compatibility	High: unmodified guest software can be executed.	High: this support both unmodified and adapted guest software.	Low: guest software has to be ported.

Most of the recent advances on virtualization have been done in desktop systems. The application of these advances to embedded systems is not a straightforward activity, due to restrictions related to determinisms, fault tolerance, hard real-time constrains, and costs, among others constraints. Therefore, in determined application domains those latest advances in hardware virtualization

extensions could not be available and for such cases, it would require to have other virtualization options such as para-virtualization.

In the context of the DREAMS project, XtratuM will be available for three architectures: X86, PowerPC and ARM. The three approaches described above have been used in the developing of XtratuM for such architectures.

In Table 1 a comparison of virtualization techniques from the point of view of the impact in the developing of hypervisors is presented. The section 2.1.3 provides additional information about this table.

2.1.1 Full virtualization

Full virtualization requires that every component and feature of the hardware architecture to be reflected into each one of multiples instances of the virtual machines. This mimicry of hardware should include the full instruction set, input/output operations, interrupts, memory access, and whatever other elements are used by the software that runs on the bare machine.

With this technique, the virtualization layer will allow multiple unmodified guest operating system instances to run concurrently within virtual machines on a single computer, dynamically partitioning and sharing the available real physical resources. However, this technique requires the development of some virtual devices that emulate real devices that can be recognized by the guest OS in the same way as on native hardware. For example, the audio hardware manufacturer on host could be Realtek and the audio virtual hardware manufacturer on guest could be Creative. It implies the development of devices firmware emulation into or as additional modules of the virtualization layer. Additionally, the hypervisor should include hardware drivers to handle real devices when virtual devices have to change the state of real I/O devices. Some operations can be executed directly in the hardware and it does not have to be emulated, such as processor, memory locations and arithmetic registers.

The feature described above increases the complexity of the development of the virtualization layer and establishes an implicit dependence of the hypervisor with the real hardware, but instead increases the portability and compatibility of “guest” software.

This virtualization technique can be achieved via classical trap-and-emulate model, using Binary Translation (BT) or using directly hardware virtualization extensions. The first could be implemented on several architectures such as SPARC. Early versions of VMWare Workstation are an example of the second and the Wind River and RTS hypervisors are an implementation of the latter.

Binary Translation translates dynamically, during the software execution, portions of the guest kernel code to replace non-virtualizable instructions with new sequences of instructions making the classic trap-and-emulate model in software possible and the execution of privileged instructions can be handled by the hypervisor, emulating its correct operation. To perform this translation the virtualization layer is developed based on an interpreter, in this way the guest software is executed on an interpreter instead of directly on a physical CPU. The interpreter separates virtual state (the VCPU) from physical state (the CPU). However, it introduces an overhead associated to the processing, execution and changing cost of instruction sets and it increases complexity in the development of the virtualization layer.

Hardware-assisted virtualization introduces additional hardware extensions to simplify virtualization techniques and provide architectural support that facilitates building the virtualization layer. The virtualization layer can virtualize the instruction set by handling privileged instructions using a classic trap-and-emulate model in hardware instead of software.

The hypervisor partitions management is simplified with more recent *hardware virtualization extensions*. With these extensions the guest state can be automatically stored in Virtual Machine

Control Structures or Virtual Machine Control Blocks, strongly reducing the code to handle context switches between partitions. Examples of virtual hardware extensions are AMD-V, Intel VT-x, AMD-Vi and Intel VT-d (Directed I/O).

2.1.2 Para-virtualization

This technique is based on an interface of communication between the guest software and the virtualization layer. The sensitive and privileged instructions are replaced by calls to the hypervisor also called “hypercalls”. This technique simplifies the building virtualization layer and improves in most cases the performance of guest software (Barham, 2003). This model is simpler to realize because only a reduced number of services are required to be implemented. Comparing the para-virtualization approach with platforms based a *trap-and-emulate* model, the para-virtualization can directly invoke a handler for each privileged instruction avoiding the code disassembling. However, compared with hardware based on the latest hardware virtualization extensions that advantage disappears. In the case of para-virtualized platforms, not only the privileged instructions can be virtualized, also a set of functionalities or instructions can be virtualized with a single hypercall.

On the other hand, the hardware used on embedded and hard real-time systems usually does not have available hardware virtualization extensions. Therefore, in most cases the para-virtualization could be the only option to implement systems based on TSP.

The main drawback of the paravirtualized system is the portability and compatibility of the “guest” software.

2.1.3 Hybrid Virtualization

This model combines the *full-virtualization* and *para-virtualization* techniques. It takes advantage of the benefits of each technique described above. This technique is mainly used to improve the performance, increase the determinism and provide additional services of communication, monitoring and supervision. Therefore, with this approach para-virtualized and unmodified guest operating systems are supported.

A comparison of virtualization techniques from the point of view of the impact in the developing of hypervisors is presented in Table 1. This table is based on the results obtained from several papers in the literature such as (Barham, 2003), (Adams, 2006), (VMware, 2007), (Guy Ben-Haim, 2012). In (Barham, 2003) a performance comparison between paravirtualized and full-virtualized hypervisors is presented. In (Adams, 2006) a comparison of software and hardware techniques on the x86 architecture is presented. In this last paper the first-generation virtualization extensions of Intel was used in the experiments. In (Guy Ben-Haim, 2012) the performance improvement using the last generation of Intel virtualization extensions are presented.

2.2 I/O Virtualization

2.2.1 IOMMU Virtualization

The Input/Output memory management unit (IOMMU) is a MMU at the bus level, i.e. the IOMMU translates physical address into virtual ones. This component allows guest partitions to directly use peripheral devices through DMA I/O bus and interrupt remapping.

Additionally, the IOMMU provides memory protection mechanisms from mischievous devices. In a traditional system the devices use DMA to access physical memory directly, therefore misbehaving or

malicious devices could read or write to any memory address. In a virtual environment the access addresses for each device can be explicitly assigned.

By using this component in the building of the hypervisor layer could also simplify the switching among partitions in a full virtualization model, allowing the use of native drivers in guest applications. The IOMMU can avoid the fail of devices that use DMA when all the memory addresses are remapped by the virtualization layer using a full virtualization approach.

Some architectures such as Intel (VT-d), AMD (AMD-Vi), SPARC (in LEON4 processors) and PowerPC (T4240 processors) have released its own version of IOMMU.

2.2.2 Device and I/O Virtualization

Depending on the virtualization technique, several I/O virtualization approaches can be used. If the technique is para-virtualization, a direct pass-through to the hardware is commonly used for device virtualization. In this case, an I/O server could be provided as an additional service of the hypervisor or as an I/O partition in order to share devices among partitions. Some devices could be reserved for an exclusive management from dedicated partitions. However, in architectures where devices cannot be allocated to a single partition in an isolated way, the implementation of an I/O server is also required. An I/O server is a dedicated partition which, on the one hand, manages several devices, and, on the other one, offers those devices to other partitions as services.

Additionally, the use of an I/O server as partition could simplify the building of the hypervisor layer and the user could improve the performance of the partition execution and, have higher control from the point of view of scheduling. For example, the user could decide when the I/O partition should be executed in order to avoid interferences with other critical partitions. The latter is especially important in mixed-criticality system.

On the other hand, as the full-virtualization technique emulates real devices as virtual devices, the devices virtualization in this technique involves managing the routing of I/O requests between virtual devices and the shared real physical hardware. This management is commonly included in the hypervisor which adds complexity to the implementation of the hypervisor although increasing the portability of the “guest” software. It also could have an impact on performance.

There are several hardware extensions for the device virtualization such as PCI-SIG I/O Virtualization, which allows to natively sharing PCI Express devices, Network Virtualization (Intel VT-c), which improves networking and I/O throughput, and Single-root I/O Virtualization (SR-IOV) that provides near native-performance by providing dedicated I/O to virtual machines. These hardware extensions reduce the software management of devices into the hypervisor. Thus, on the one hand, improving the performance of the system, as well as improving data isolation among virtual machines, and therefore providing flexibility and mobility. This is because the hardware extensions bypass the software virtual switch in the hypervisor. However, these hardware virtualization extensions are currently available in a reduced number of systems.

2.3 Guest Operating System

From the point of view of the Guest Operating systems, several factors must be deeply analysed before selecting the virtualization technique to be used:

- Source code availability. This a crucial factor, since para-virtualization depends on the replacement of low-level functionalities by high-level ones.
- System performance and determinism. The application requirements of the application, such as real-time constraints, best-effort performance, or safety, among others could determine the most suitable approach. In general, in most cases the hybrid-virtualization will be the best option.

- Portability and compatibility. A full virtualized environment enables the direct execution of an unmodified system, easing portability. Additionally, another advantage to use operating systems on a full virtualized machine is that it can be used to provide a common, more generic underlying virtual hardware, regardless of the real one.

3 XtratuM hypervisor overview

XtratuM is a real-time hypervisor that it is intended for execution in a bare computer. XtratuM shares out the resources (i.e. memory and computing time) between the Partitions. A partition is an independent execution unit designed to execute under XtratuM's control.

XtratuM is a hypervisor that can include para-virtualization, full-virtualization or the combination of both techniques to build a virtualization layer. The techniques used will depend on the hardware support.

Basically, the hypervisor provides multiple isolated virtual machines, or partitions. Each partition can execute a complete system (i.e. OS kernel and the application processes) or bare applications (i.e. execution runtime and application). Communication between partitions is done commonly by means of a virtual network or inter-partition communications mechanisms. From the application layer, a hypervisor system is much more than a distributed system: a set of computers, where each computer runs its own operating system and applications, and computers are inter-connected with a high speed network. The most noticeable difference is the speed of the virtual computers, which is only a fraction of the native computer.

Figure 1 shows the approach in moncore. The hypervisor virtualizes the CPU and offers a virtual CPU to the partitions.

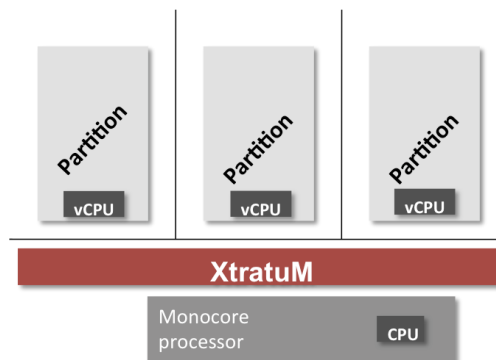


Figure 1: Monocore virtualization

In a multicore approach, the hypervisor can provide several virtual CPUs to the partitions. A partition can be mono or multicore. Different partitions (from the point of view of the number of cores) can coexist in the system. It allows to profit from a multicore platform even if the partitions are not multicore. Figure 2 shows an example of this view. It shows a multicore platform virtualized by XtratuM which offers the possibility to build multicore or monocore partitions. In the example, two partitions use all the virtualized CPUs due to it uses a multicore OS. The third partition is monocore and only uses a virtual CPU.

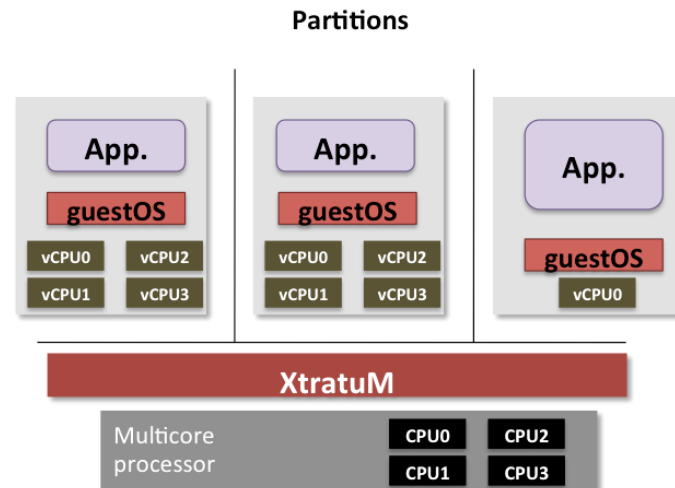


Figure 2: Multicore approach

XtratuM offers a complete hardware abstraction to allow to the partitions to execute its code as native machine. However, in a para-virtualized approach, the partitions will require to use the services provided by the hypervisor to use the virtualized resources via *hypercalls*.

In general, in multicore architectures, some sources of indeterminism can strongly impact in the WCET determination of the partitions. These sources of indeterminism are:

- Internal architecture aspects
- Cache management (L2 and L3 caches mainly)
- Shared memory accesses
- Shared controller units
- Bus arbitration

These issues can add unpredictability to the execution of the partitions. For instance, there is a problem when several cores running at the same time access to the memory. It is a source of indeterminism that is well known problem in multicore systems. In the case of a hypervisor, the problem still exists and has to be solved at partition level by performing a more complex estimation time required for the partition. From the hypervisor point of view, this temporal interference can only be avoided if the hardware base provides mechanisms to handle it. Therefore, it is not the responsibility of the hypervisor to provide a solution for that problem. In some cases, the virtualization layer could mitigate the problem by constraining the use of the resources. For instance, when cache management is available in the hardware, it could force a partitioning of the cache or other techniques. DREAMS studies how these interferences could be better mastered on current architectures in WP2 and WP4, by providing system services as the Local Resource Manager (LRM). These services will work as Xtratum extensions.

3.1 Basic properties

The basic properties to be achieved by a hypervisor for multicore mixed-critically embedded applications are:

- **Spatial isolation:** A partition is completely allocated in a unique address space (code, data, stack). This address space is not accessible by other partitions. The hypervisor has to guarantee the spatial

isolation of the partitions. The system architect can relax this property by defining specific shared memory areas between partitions. This spatial isolation is achieved using support hardware.

- **Temporal isolation:** A partition is executed independently of the execution of other partitions. In other words, the execution of a partition cannot be disturbed by the execution of other partitions. It influences directly on the scheduling policies at hypervisor level. The hypervisor has to schedule partitions under a scheduling policy that guarantees the partition execution. However, in multicore systems, the temporal interferences in parallel executions only can be avoided if the hardware base provides mechanisms to achieve it. Otherwise, the hypervisor side a time separation of system partitions is achieved by an adequate scheduling plan. The scheduling must provide an execution plan, such that no partition interferes with any other partition. The safest scheduling policy provided is static cyclic scheduling, where partition execution times are configured a priori and cannot be changed during an execution. The hypervisor can model the hardware temporal interferences using different scheduling algorithms. Using this method the effect of the interferences can be significantly reduced, but they cannot be disabled.
- **Predictability:** A partition with real-time constraints has to execute its code in a predictable way. It can be influenced by the underlying layers of software (guest-OS and hypervisor) and by the hardware. From the hypervisor point of view, the predictability applies to the provided services, the operations involved in the partition execution and the interruption management of the partitions.
- **Security:** All the information in a system (partitioned system) has to be protected against access and modification from unauthorised partitions or unplanned actions. Security implies the definition of a set of elements and mechanisms that permit to establish the system security functions. This property is strongly related with the static resource allocation and a fault model to identify and confine the vulnerabilities of the system.
 - **Static resource allocation:** The system architect is the responsible of the system definition and resource allocation. This system definition is detailed in the configuration file of the system specifying all system resources, namely, number of CPUs, memory layout, peripherals, partitions, the execution plan of each CPU, etc. Each partition has to specify the memory regions, communication ports, temporal requirements and other resources that are needed to execute the partition code. Static resource allocation is the basis of predictability and safety of the system. The hypervisor has to guarantee that a partition can access to the allocated resources and deny the requests to other not allocated resources.
 - **Fault isolation and management:** A fundamental issue in critical systems is the fault management. Faults, when occur, have to be detected and handled properly in order to isolate them and avoid the propagation. A fault model to deal with the different types of errors is to be designed. The hypervisor has to implement the fault management model and allowing the partitions to manage those errors that involve the partition execution.
- **Partition support:** The execution environments are required to be adapted to work on a virtual platform. The hypervisor has to provide the support to execute partitions and inform about how the system is working.
- **Confidentiality:** Partitions cannot access to the space of other partitions neither see how the system is working. From their point of view, they only can see their own partition. This property

can be relaxed to some specific partitions in order to see the status of other partitions or control their execution.

4 XtratuM – Software design overview

This section is focused mainly on the XtratuM hypervisor using a para-virtualized approach for the DREAMS harmonized platform. This platform consists of a Zynq-7000 Board using an processor ARM Cortex A9, which does not incorporate hardware virtualization extensions. However, many services described below will be used in a similar way on XtratuM using others virtualization approaches. Additionally, the execution environment of the partition will be assumed as a bare partition using a minimal runtime to allow the execution of applications. This runtime could be the minimal execution runtime provided by XtratuM and called XAL, or it could be a minimal Bare-C Cross-compiler system around GNU/GCC tools.

4.1 Software static architecture

4.1.1 System deployment

XtratuM is a bare metal hypervisor intended mainly for embedded real-time systems. Figure 3 shows the expected deployed system: the XtratuM hypervisor runs on a specific board (e.g. Zynq-7000 Board, ATOM-Intel Board, T4240-QDS PPC board), managing it and providing multiple virtual execution environments (i.e. partitions). The application, XAL/DRAL and the LibXM (library provided by XtratuM) are executed within one of these partitions.

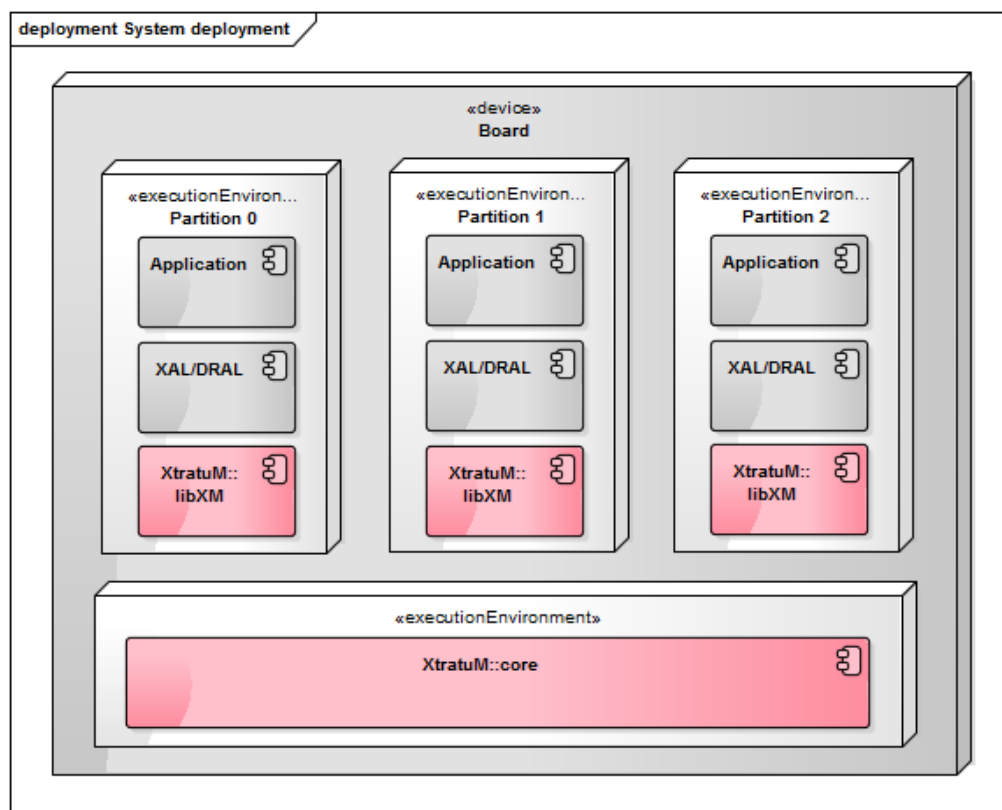


Figure 3: System deployment diagram

4.1.2 System components

The system (Figure 4) is composed by (bottom to top, external components in gray) the Hardware Layer, the XtratuM hypervisor, XAL/DRAL and the user application.

Zynq-7000 Board: Hardware platform [7] where the software components are executed. It includes a dual-core 32-bit ARM Cortex-A9 processor, OCM, SDRAM and FLASH. Software components interact with this component through the processor registers and the memory ports. The Cortex-A9 processor has available TrustZone technology[8], which allows the execution of software in two different worlds: a) *Secure domain*, it has access to all instructions and resources on the system; b) *Normal domain*, it has the same capabilities as secure domain from the point of view of execution and access to processor (including all processor modes), but in this mode only the resources¹ previously allocated by the secure domain are available.

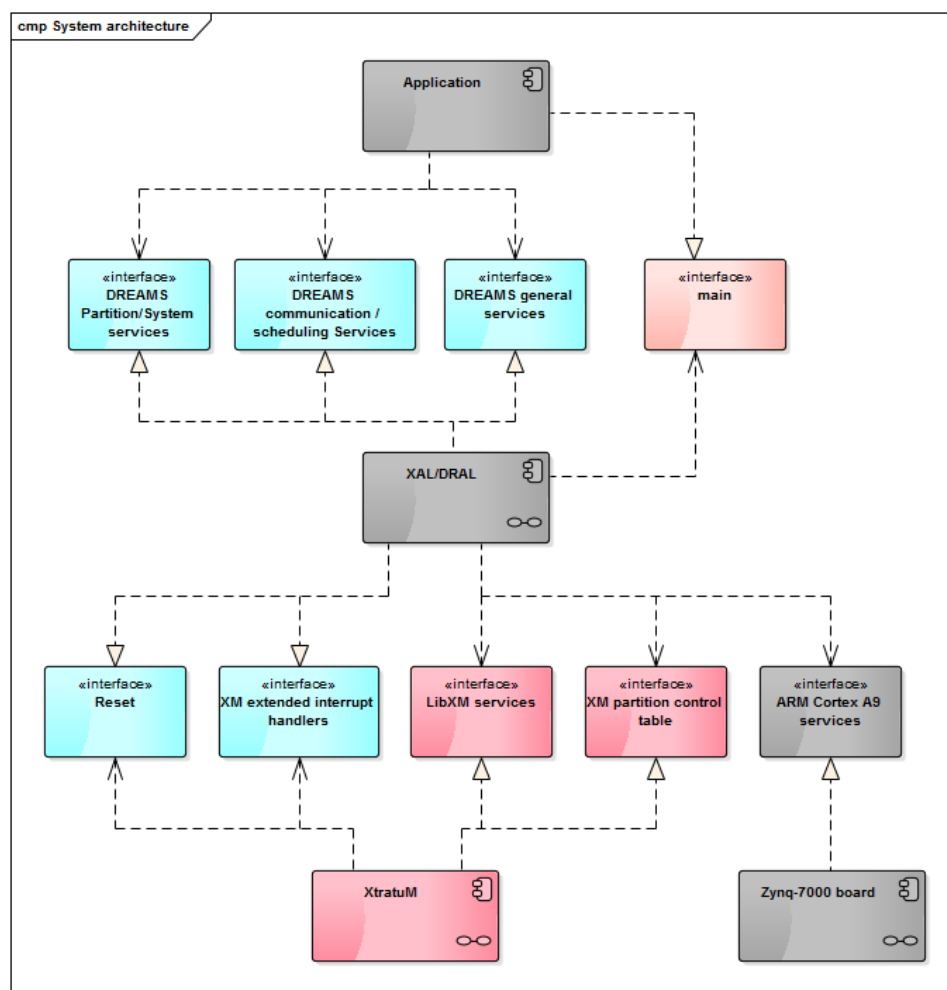


Figure 4: System deployment diagram

XtratuM hypervisor: XtratuM hypervisor [8] manages the underlying hardware, providing multiple virtual execution environments. For each of these virtual execution environments, the hypervisor allocates a set of physical resources to be used directly by the partition and implements a set of virtual devices, such as timer and clock, behaving similarly to their hardware counterparts. Additionally, the hypervisor provides a subset of custom services and

¹ Resources refer to physical memory, devices management, interrupts, coprocessors, etc.

functionality related to management, monitoring and control at the partition and system levels. Inter-partition, inter-tile and inter-node communication services are also provided by the hypervisor via the DRAL layer. The interaction of the software with these services is performed through a set of low-level services provided by the hypervisor in the LibXM. The hypervisor is released jointly with the LibXM that may be linked jointly with the partition code providing a C interface to the XtratuM services. The use of this library is not compulsory, nonetheless, the partition may be in charge of implementing the hypervisor service invocation convention. The virtual environments run in “*Normal domain*” such as defined by ARM TrustZone technology [9]. While the hypervisor runs in “*Secure domain*” in order to manage the access to the physical resources that can be performed from the partitions.

XAL/DRAL: XAL is a minimal run-time environment provided by XtratuM to execute simple partition code. XAL provides a developing environment to create bare “C” applications. XAL is provided jointly with the XtratuM sources as a library “LibXAL”. This environment is linked by default with the library LibXM and it can be used as tool to test the XtratuM services.

DRAL is the *DREAMS Abstraction Layer*. This layer includes specific DREAMS services involving hypervisor, partition, scheduling and communication services. DRAL offers a homogenous interface to the applications to access low level services in the DREAMS platform. This software component will be addressed in section 0.

Application: Software payload executed within a virtual execution environment provided by XtratuM. The application is linked with the DRAL library, LibXAL and LibXM. It uses the services and abstractions provided by DRAL and XAL.

Note that the application could use a more complex operative system instead of a minimal run-time. For such case, XAL would be replaced by the new O.S. but DRAL maintains the same application interface, although it would be a DRAL adapted to such O.S.

4.1.2.1 Zynq-7000 board

In the specific case for the DREAMS harmonized platform, it uses a Zynq-7000 series board, which incorporates the ARM Cortex-A9 Dual core as processing system (PS). The Cortex-A9 is a 32 bit processor core that implements the ARMv7A architecture.

This processor implements the ARM TrustZone technology and XtratuM takes advantage of this technology to perform virtualization of the system. The aim of ARM’s Trustzone technology was not to make the ISA (Instruction Set Architectures) virtualizable, but to increase system security. However, it still exhibits some useful properties that can help virtualization.

Trustzone introduces two new modes called worlds, *secure domain* and *normal domain*. Software executing in the normal world cannot access resources belonging to the secure world. Secure world software on the other hand can access non secure resources. All exception and processor modes are available in both worlds. Interrupts and devices can be assigned to either world. If they are assigned to the secure world it is possible to expose them to normal world software, but only via using defined interfaces which invoke secure world software. For example all interrupts are handled by secure world software and are only forwarded directly to the normal world when configured accordingly. Also devices that are allocated to the secure world can only, if an interface is exposed to them, be accessed indirectly by normal world software. However, the devices can also be allocated directly to normal world if it is configured in that way. This indirection prevents normal world (the non-secure world) software from interfering with the secure world.

As those examples show TrustZone allows aspects of the normal world to be managed by the secure mode using predefined allocation of resources.

4.1.2.2 XtratuM top-level architecture

Figure 5 shows the decomposition of XtratuM in its major sub-components as well as its interface with DRAL. These sub-components are (from top to bottom):

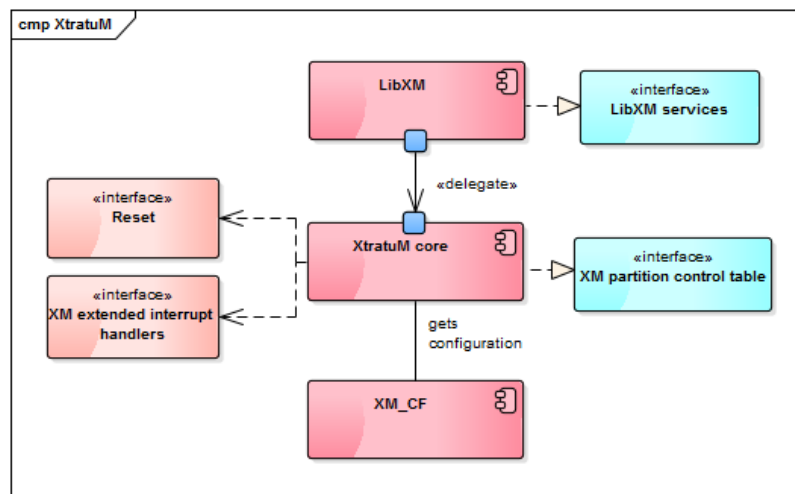


Figure 5: XtratuM top-level architecture

LibXM: Library released jointly with XtratuM source code. This library may be optionally linked with the partition software providing a C interface to invoke XtratuM services. It is used by XAL/DRAL to interface with XtratuM.

Application: Software payload executed within a virtual execution environment (partition) provided by XtratuM. The application is linked with the DRAL library, LibXAL and LibXM. It uses the services and abstractions provided by DRAL and XAL.

XtratuM core: The core manages the underlying hardware, providing the abstraction and facilities required to run multiple virtual execution environments on the hardware. For each of these virtual execution environments, the hypervisor provides a set of services that could be classified as: system and partition management services, time and scheduling, communications, health-monitoring and tracing services, virtual extended interrupt management and multicore support. Additionally, the hypervisor can offer virtual devices mimicking the ones present in underlying hardware when several execution environments require the same physical device. These services and virtual devices are managed through a set of low-level services implemented by the core. Each partition includes a partition control table. The partition control table is a memory area shared between the partition and XtratuM, used by this last one to provide relevant information. Partitions are scheduled according to a cyclic scheduling policy. Although, the cyclic scheduling is the policy typically used to schedule partitions in critical systems, there are other scheduling policies available to be used with XtratuM.

XM_CF (XtratuM Configuration file): This component is a table defining the hypervisor configuration, among other things, partitions, inter-partition communication, channels, resource allocation: memory maps, interrupts, devices, etc. The XtratuM Configuration File (XM_CF) is the translation of the XML file used to configure the hypervisor into a binary format understandable by the hypervisor.

4.1.3 System states

XtratuM is a software component that shares out the memory and computes time between a set of partitions following a configuration plan. To be able to share the resources, XtratuM needs to initialize a set of data, and load each partition in memory. This is done at first in the **BOOT** state, that includes the period of time between starting from the entry point, to the execution of the first partition. In this state the scheduler is not enabled and the partitions are not executed.

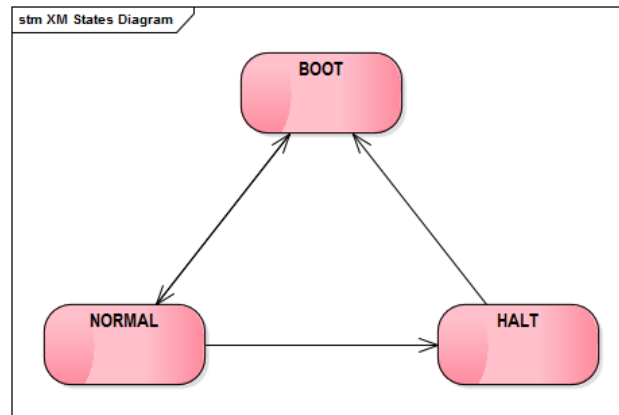


Figure 6: System states diagram

At the end of the boot sequence the hypervisor is ready to start executing partition code. The system is in **NORMAL** state and the scheduling plan is started. XtratuM only runs when a service is requested by the current partition, or when an asynchronous event arrives.

The system can switch to **HALT** state by the health monitor system in response to a detected error or by the *system partition* invoking the halt system hypercall. In the halt state the scheduler is disabled, the hardware interrupts are disabled, and the processor enters in a power sleep mode or in an endless loop (configuration dependent). The only way to exit from this state is via an external hardware reset.

4.2 Partition overview

A partition is an independent execution unit designed to execute under XtratuM's control. Each partition can execute a complete system (OS kernel and the application processes) or bare applications (execution runtime and application). The latter is independent from the point of view of the hypervisor.

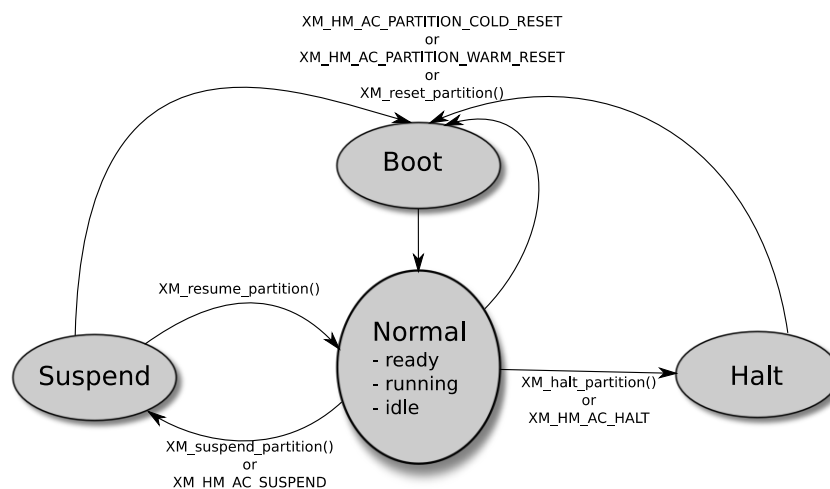


Figure 7: Partition states and transitions

4.2.1 Partition operation

Once XtratuM is in normal state, partitions are started. The partition's states and transitions are shown in Figure 7.

On start-up each partition is in boot state. XtratuM has to prepare the virtual machine to be able to run the applications: it sets up a standard execution environment (that is, initializes a correct stack and sets up the virtual processor control registers), creates the communication ports, requests the hardware devices (I/O ports and interrupt lines), etc., that it will use. Once the partition has been initialized, it changes to normal mode.

The partition receives information from XtratuM about the previous executions, if any.

From the hypervisor point of view, there is no difference between the boot state and the normal state.

In both states the partition is scheduled according to the fixed plan, and has the same capabilities. Although not mandatory, it is recommended that the partition emits a partition's state-change event when changing from boot to normal state.

The normal state is subdivided in three sub-states:

- **Ready.** The partition is ready to execute code, but it is not scheduled because it is not in its time slot.
- **Running.** The partition is being executed by the processor.
- **Idle.** If the partition does not want to use the processor during its allocated time slot, it can relinquish the processor and wait for an interrupt or for the next time slot (see `XM_idle_self()`).

A partition can halt itself or be halted by a system partition. In the halt state, the partition is not selected by the scheduler and the time slot allocated to it is left idle (it is not allocated to other partitions). All resources allocated to the partition are released. It is not possible to return to normal state.

In suspended state, a partition will not be scheduled and interrupts are not delivered. Interrupts raised while in suspended state are left pending. If the partition returns to normal state, then pending interrupts are delivered to the partition. The partition can return to ready state if requested by a system partition by calling `XM_resume_partition()` hypercall.

4.2.2 Types of partitions

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.

Note that system partition 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 configuration file.

A partition has system capabilities if the `/System_Description/Partition_Table/Partition/@flags` attribute contains the flag "*system*" in the XML configuration file. Several partitions can be defined as system partition.

4.2.3 Names and identifiers

Each partition is globally identified by a unique identifier *id*. Partition identifiers are assigned by the integrator in the XM CF file. XtratuM uses this identifier to refer to partitions. System partitions use partition identifiers to refer to the target partition. The “C” macro *XM_PARTITION_SELF* can be used by a partition to refer to itself.

These *ids* are used internally as indexes to the corresponding data structures. The first identifier (*id*) of each object group shall start in zero and the next *id*’s shall be consecutive. It is mandatory to follow this order in the XM_CF file.

The attribute *name* of a partition is a human readable string. This string shall contain only the following set of characters: upper and lower case letters, numbers and the underscore symbol. It is advisable not to use the same name on different partitions. A system partition can get the name of another partition by consulting the status object of the target partition.

In order to avoid name collisions, all the hypercalls of XtratuM contain the prefix “XM”. Therefore, the prefix “XM”, both in upper and lower case, is reserved.

4.3 Partition scheduling

XtratuM main schedules partitions in a fixed, cyclic basis (ARINC-653 scheduling policy). This policy ensures that one partition cannot use the processor for longer than scheduled to the detriment of the other partitions. The set of *time slots* allocated to each partition is defined in the XM_CF configuration file during the design phase. Each partition is scheduled for a *time slot* defined as a start time and a duration. Within a *time slot*, XtratuM allocates the processor to the partition.

If there are several concurrent activities in the partition, the partition shall implement its own scheduling algorithm. This two-level scheduling scheme is known as hierarchical scheduling. XtratuM is not aware of the scheduling policy used internally on each partition.

In general, a cyclic plan consists of a *major time frame* (MAF) which is periodically repeated. The MAF is defined as the least common multiple of the periods of the partitions (or the periods of the threads of each partition, if any).

Other scheduling schemas of partitions are available to be used on XtratuM. Some of these schemas are based on fixed-priorities and cyclic with *spare* capabilities. Scheduling schemas as results of DREAMS WP3 could be included as scheduling schemas of partitions in XtratuM.

4.3.1 Multiple scheduling plans

Using the cyclic scheduling schema to schedule partitions, in some cases, a single scheduling plan may be too restrictive. For example:

- Depending on the guest operating system, the initialisation can require a certain amount of time and can vary significantly. If there is a single plan, the initialisation of each partition can require a different number of slots due to the fact that the slot duration has been designed considering the operational mode. This implies that a partition can be executing operational work whereas others are still initialising its data.
- The system can require to execute some maintenance operations. These operations can require allocating other resources different from the ones required during the operational mode.

In order to deal with these issues, XtratuM provides multiple scheduling plans that allow reallocating the timing resources (i.e. the processor) in a controlled way. In the scheduling theory this process is known as mode changes. Figure 8 shows how the modes have been considered in the XtratuM scheduling.

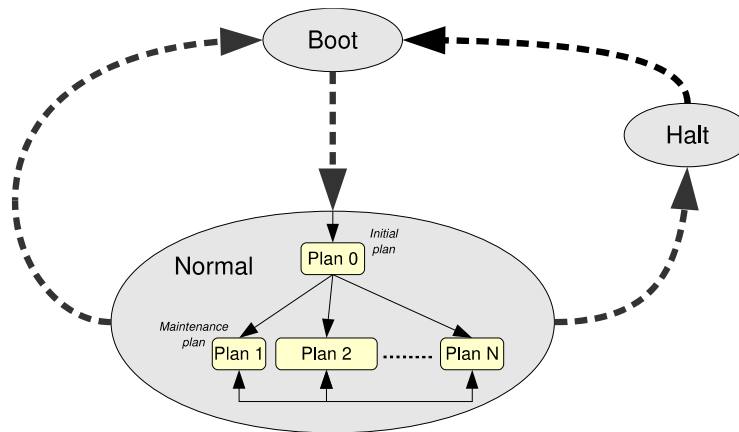


Figure 8: Scheduling modes

The scheduler (and so, the plans) is only active while the system is in normal mode. Plans are defined in the XM_CF file and identified by a identifier. Some plans are reserved or have a special meaning:

Plan 0: *Initial plan.* The system selects this plan after a system reset. The system will be in plan 0 until a plan change is requested.

Plan 1: *Maintenance plan.* This plan can be activated in two ways:

- As a result of the health monitoring action XM_HM_AC_SWITCH_TO_MAINTENANCE. The plan switch is done immediately.
- Requested by a system partition. The plan switch occurs at the end the current plan.

It is advisable to allocate the first slot of this plan to a system partition, in order to start the maintenance activity as soon as possible after the plan switch. Once the maintenance activities have been completed, it is responsibility of a system partition to switch to another plan (if needed).

A system partition can also request a switch to this plan.

Plan x (x>1): Any plan greater than 1 is user defined. A system partition can switch to any of these defined plans at any time.

4.4 Inter-Partition communications (IPC)

Inter-partition communications are communications between two or more partitions. XtratuM implements a message passing model which highly resembles the one defined in the ARINC-653 standard. A message is a variable block of data. A message is sent from a source partition to one or more destination partitions. The data of a message is transparent to the message passing system.

A communication channel is the logical path between one source and one or more destinations. Partitions can access to channels through access points named ports. The hypervisor is responsible for encapsulating and transporting messages that have to arrive to the destination(s) unchanged. At partition level, messages are atomic entities: either the whole message is received or nothing is received. Partition developers are responsible for agreeing on the format (data types, endianness, padding, etc.).

XtratuM provides two basic transfer modes: *sampling* and *queuing*.

Channels, ports, maximum message sizes and maximum number of messages (queuing ports) are entirely defined in the configuration files.

Sampling port: It provides support for broadcast, multicast and unicast messages. No queuing is supported in this mode. A message remains in the source port until it is transmitted through the channel or it is overwritten by a new occurrence of the message, whatever occurs first. Each new

instance of a message overwrites the current message when it reaches a destination port, and remains there until it is overwritten. This allows the destination partitions to access the latest message.

The channel has an optional configuration attribute named `@refreshPeriod`. This attribute defines the maximum time that the data written in the channel is considered “valid”. Messages older than the valid period are marked as invalid. When a message is read, a bit is set accordingly to the valid state of the message.

Queuing port: It provides support for buffered unicast communication between partitions. Each port has a queue associated where messages are buffered until they are delivered to the destination partition. Messages are delivered in FIFO order.

If the requested operation cannot be completed because the buffer is full (when trying to send a message) or empty (when attempting to receive a message), then the operation returns immediately with the corresponding error. The partition’s code is responsible for retrying the operation later.

In order to optimise partition’s resources and reduce the performance loss caused by polling the state of the port, XtratuM triggers an extended interrupt when a new message is written/sent to a port. Since there is only one single interrupt line to notify for incoming messages, on the reception of the interrupt, the partition code has to determine which port or ports are ready to perform the operation. XtratuM maintains a bitmap in the Partition Control Table to inform about the state of each port. A “1” in the corresponding entry indicates that the requested operation can be performed.

When a new message is available in the channel, XtratuM triggers an extended interrupt to the destination(s).

4.5 Health Monitor (HM)

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

It is important to clearly understand the difference between 1) an incorrect operation (instruction, function, application, peripheral, etc.) which is handled by the normal control flow of the software, and 2) an incorrect behaviour which affects the normal flow of control in a way not considered by the developer or which can not be handled in the current scope.

An example of the first kind of errors is: the `malloc()` function returns a null pointer because there is not memory enough to attend the request. This error is typically handled by the program by checking the return value. As for the second kind, an attempt to execute an undefined instruction (processor instruction) may not be properly handled by a program that attempted to execute it.

The XtratuM health monitoring system will manage those faults that cannot, or should not, be managed at the scope where the fault occurs.

The XtratuM HM system is composed of four logical blocks:

- **HM event detection:** to detect abnormal states, using logical probes in the XtratuM code.
- **HM actions:** a set of predefined actions to recover from the fault or confine an error.
- **HM configuration:** to bind the occurrence of each HM event with the appropriate HM action.
- **HM notification:** to report the occurrence of the HM events.

Note that only some events are detected directly by the hypervisor. Most hardware events are received directly by the partitions that generate them. Partition events must be handled by the partitions and those events are not detected by XtratuM. Therefore, the hypervisor provides to the partition a mechanism for the notification of events, which will be processed based on actions

configured in the configuration file. These services of notification should be used by the partitions in order to maintain the scheme proposed by the hypervisor XtratuM to detect and react to anomalous states.

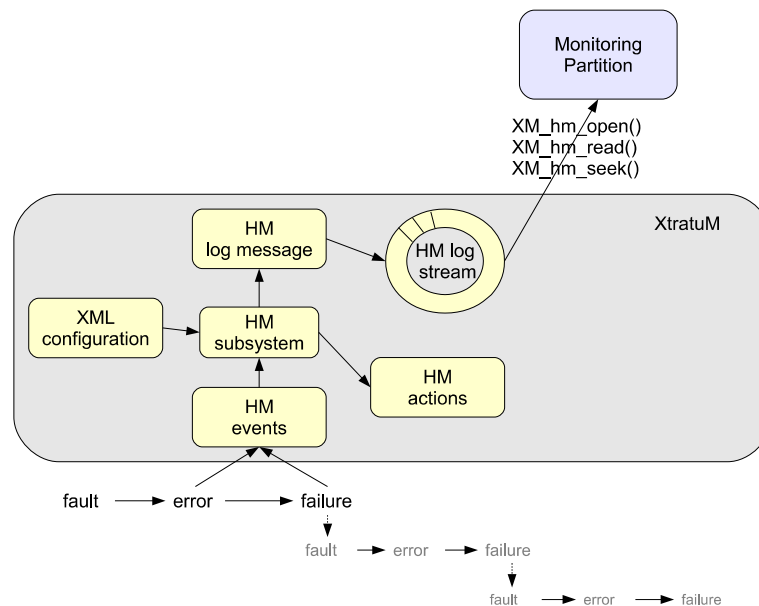


Figure 9: Health monitoring overview.

Once the defined HM action is carried out by XtratuM, a HM notification message is stored in the HM log stream (if the HM event is marked to generate a log). A system partition can then read those log messages and perform a more advanced error handling. In Figure 9 a health monitoring overview is shown.

4.6 Inter-Partition Virtual Interrupts (IPVI)

An Inter-Partition Virtual Interrupt (IPVI) emulates the way a real Inter-Processor Interrupts (IPIs) works in real processors. That is, every time the correspondent hypercall is invoked, a virtual interrupt is caused to a destination partition. An IPVI can be raised by any partition.

Each partition has a maximum of 8 IPVIs, implemented as the last eight extended virtual interrupts. The system integrator, through the XML, indicates the entity who receives a IPVI after being raised.

<Channels>

<lpvi id="0" sourceId="5" destinationId="8" />

<lpvi id="0" sourceId="4" destinationId="1" />

<lpvi id="1" sourceId="4" destinationId="8, 1" />

</Channels>

The example above shows a configuration where the behaviour of three IPVIs is defined: the IPVI 0, caused by the partition 5 and received by the partition 8. The IPVI 0 and 1, caused both by the partition 4 and received, the first one by the partition 1 and the second by the partitions 8 and 1.

4.7 Interfaces context required by XtratuM

Xtratum requires a set of hardware services that are strongly dependent on the processor. The following type of interfaces are required for the ARM Cortex-A9 processor:

Interface	Required
CPU Registers	Partition context switch.
Timers	Timer handler and external interface. Provide a system time.
TrustZone interface	Control of execution worlds. Allocation of memory, interrupts, devices, coprocessor access, among others.
MMU	Memory management.
Interrupt device	Interrupt handler and management. Virtualize hardware interrupts when these are shared, otherwise the interrupts are allocated directly to the partition and managed by the partition.
Cache control unit	Cache management
IO Ports/devices	Device management and external interface

4.8 Interfaces context provided by XtratuM

4.8.1 Hypercalls

The external interface is provided through the libXM that provides to the partition the XM-API. The exact list of services provided by XtratuM is detailed in the [10] Manual. Next tables list a subset of these services that should be taken as example.

4.8.1.1 System services

Interface	Required
XM-System services	Services related to the system management
XM_get_system_status	Get the current status of the system.
XM_halt_system	Stop the system.
XM_reset_system	Reset the system.

4.8.1.2 Partitioning services

Interface	Required
XM-Partitioning services	Services related to the partition management
XM_get_partition_status	Get the current status of a partition.
XM_halt_partition	Terminates a partition.

XM_params_get_PCT	Gets the address of the Partition Control Table
XM_reset_partition	Reset a partition.
XM_resume_partition	Resume a partition
XM_set_partition_opmode	Informs the internal status of the partition
XM_shutdown_partition	Send a shutdown interrupt to a partition.
XM_suspend_partition	Suspend the execution of a partition.

4.8.1.3 Time services

Interface	Required
XM-Time services	Services related to the time management
XM_get_time	Gets the global or local time
XM_set_timer	Arm a timer based on a global or local time

4.8.1.4 Plan schedule services

Interface	Required
XM-Plan services	Services related to the plan management
XM_get_plan_status	Return information about the scheduling plans.
XM_switch_sched_plan	Request a plan switch at the end of the current MAF.

4.8.1.5 Inter-partition communication services

Interface	Required
XM-IPC services	Services related to the inter-partition communication
XM_create_queuing_port	Create a queuing port.
XM_create_sampling_port	Create a sampling port
XM_get_queuing_port_info	Get the info of a queuing port.
XM_get_queuing_port_status	Get the status of a queuing port.
XM_get_sampling_port_info	Get the info of a sampling port.
XM_get_sampling_port_status	Get the status of a sampling port.
XM_read_sampling_message	Read a message from the specified sampling port
XM_receive_queuing_message	Receive a message from the specified queuing port.
XM_send_queuing_message	Send a message in the specified queuing port.

XM_write_sampling_message	Writes a message in the specified sampling port.
---------------------------	--

4.8.1.6 Health monitor services

Interface	Required
XM-HM services	Services related to the HM management
XM_hm_get_app_error	Read a application health monitoring log entry.
XM_hm_raise_app_error	Raises an application error
XM_hm_read	Read a health monitoring log entry.
XM_hm_seek	Sets the read position in the health monitoring stream.
XM_hm_status	Get the status of the health monitoring log stream.

4.8.1.7 Tracing services

Interface	Required
XM-Trace services	Services related to the Trace management
XM_trace_event	Records a trace entry.
XM_trace_open	Open a trace stream.
XM_trace_read	Read a trace event.
XM_trace_seek	Sets the read position in a trace stream.
XM_trace_status	Get the status of a trace stream.

4.8.1.8 Interrupt management services

Interface	Required
XM-IRQ services	Services related to the Virtual Interrupt management
XM_clear_irqmask	Unmask virtual interrupts.
XM_clear_irqpend	Clear pending virtual interrupts.
XM_route_irq	Link an virtual interrupt to a specific interrupt vector
XM_set_irqmask	Mask virtual interrupts.
XM_set_irqpend	Force some virtual interrupts as pending.

4.8.2 Binary interfaces

This section covers the data types and the format of the files and data structures used by XtratuM.

XtratuM conforms to the following conventions of basic data types:

Unsigned	Signed	Size (bytes)	Alignment
xm_u8_t	xm_s8_t	1	1
xm_u16_t	xm_s16_t	2	4
xm_u32_t	xm_s32_t	4	4
xm_u64_t	xm_s64_t	8	8

These data types have to be stored in the endianness format required according to the underlying architecture,.

“C” declaration which meet these definitions are presented in the list below:

```
// Basic types
typedef unsigned char xm_u8_t;
typedef char xm_s8_t;
typedef unsigned short xm_u16_t;
typedef short xm_s16_t;
typedef unsigned int xm_u32_t;
typedef int xm_s32_t;
typedef unsigned long long xm_u64_t;
typedef long long xm_s64_t;

// Extended types
typedef long xmLong_t;
typedef xm_u32_t xmWord_t;
#define XM_LOG2_WORD_SZ 5
typedef xm_s64_t xmTime_t;
#define MAX_XMTIME 0x7fffffffffffffffffLL
typedef xm_u32_t xmAddress_t;
typedef xmAddress_t xmIoAddress_t;
typedef xm_u32_t xmSize_t;
typedef xm_s32_t xmSSize_t;
typedef xm_u32_t xmId_t;
```

For future compatibility, most data structures contain version information. It is a xm_u32_t data type with 3 fields: version, subversion and revision. The macros listed next can be used to manipulate those fields:

```
#define XM_SET_VERSION(_ver, _subver, _rev) ((((_ver)&0xFF)<<16)|(((  
_subver)&0xFF)<<8)|((_rev)&0xFF))  
#define XM_GET_VERSION(_v) (((_v)>>16)&0xFF)  
#define XM_GET_SUBVERSION(_v) (((_v)>>8)&0xFF)  
4.8.3 #define XM_GET_REVISION(_v) ((_v)&0xFF)
```

4.8.4 Partition control table (PCT)

In order to minimize the overhead of the para-virtualized services, XtratuM defines a special data structure that is shared between the hypervisor and the partition called Partition control table (PCT). There is a PCT for each partition. XtratuM uses the PCT to send relevant operating information to the partitions. The PCT is mapped as read-only, allowing a partition only to read it. Any write access causes a system exception. Partitions can access this table using the address provided by the `XM_params_get_PCT` macro.

```
typedef struct {
    xm_u32_t magic;
    xm_u32_t xmVersion; // XM version
    xm_u32_t xmAbiVersion; // XM's abi version
    xm_u32_t xmApiVersion; // XM's api version
    xm_u32_t resetCounter; // Number of partition reset
    xm_u32_t resetStatus; // Reset status
    xm_u32_t cpuKhz; // CPU frequency
    xmId_t id; // Partition identifier
    // Copy of kthread->ctrl.flags
    xm_u32_t flags;
    xm_u32_t imgStart;
    xm_u32_t hwIrqs[CONFIG_NO_HWIRQS/32]; // Hw interrupts belonging
    to the partition
    xm_s32_t noPhysicalMemAreas; // No of memory areas
    xm_s32_t noCommPorts; // No of comm. ports
    xm_u8_t name[CONFIG_ID_STRING_LENGTH];
    xm_u32_t iFlags;
    xm_u32_t hwIrqsPend[CONFIG_NO_HWIRQS/32]; // pending hw irqs
    xm_u32_t hwIrqsMask[CONFIG_NO_HWIRQS/32]; // masked hw irqs

    xm_u32_t extIrqsPend; // pending extended irqs
    xm_u32_t extIrqsMask; // masked extended irqs

    struct pctArch arch;
    struct {
        xm_u32_t noSlot:16, releasePoint:1, reserved:15;
        xm_u32_t id;
        xm_u32_t slotDuration;
    } schedInfo;
    xm_u16_t trap2Vector[NO_TRAPS];
    xm_u16_t hwIrq2Vector[CONFIG_NO_HWIRQS];
    xm_u16_t extIrq2Vector[XM_VT_EXT_MAX];
} partitionControlTable_t;
```

4.8.5 Virtual Interrupts

4.8.5.1 Interrupt model

Different manufacturers use terms like exceptions, faults, aborts, traps, and interrupts to describe the processor mechanism to receive a signal indicating the need for attention. Also, different authors adopt different terms to their own use. In order to define the interrupt model, we provide the generic definition of the terms used in this work.

A trap is the mechanism provided by the processor to implement the asynchronous/synchronous transfer of control. When a trap occurs, the processor switches to a privileged mode and unconditionally jumps into a predefined handler.

A software trap is raised by a processor instruction and it is commonly used to implement the system call mechanism in the operating systems.

An exception is an automatically generated interrupt that occurs in response to some exceptional condition violation. It is raised by the processor to inform about a condition that prevents the continuation of the normal execution sequence.

A hardware interrupt is a trap raised due to an external hardware event (external to the CPU). These interrupts generally have nothing at all to do with the instructions currently executing and informs the CPU that a device needs some attention.

In a partitioned system, the hypervisor can handle these interrupts (native interrupts) and generate the appropriated virtual interrupts to the partitions. However, if the hardware features allow it, the hypervisor can delegate the management of these native interrupts directly to the partition. In such case, the interrupts are caught directly by the partition and the hypervisor does not take part in the handle, only in the allocation of such interrupt to the partition. In a general way, a partition has to deal with the following virtual traps:

- **Virtual exceptions** are the exceptions propagated by the hypervisor to the partitions as consequence of a *native exception* occurrence. Not all the *native exceptions* are propagated to the partition. 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 (like memory isolation fault).
- **Native exceptions** are the exceptions handled directly by the partition due to the assignment done from the hypervisor.
- **Virtual hardware interrupts** are interrupts generated by the real hardware, handled by the hypervisor and afterwards propagated by the hypervisor to the partitions as virtual interrupts. The hardware interrupts correspond to the signals generated from external devices (dedicated devices technique) or peripherals.
- **Native hardware interrupts** are the hardware interrupts directly allocated to the partition to be handled directly by them. These interrupts are raised to the partition in the same way as in a bare-metal machine.
- **Virtual Extended interrupts** or *virtual partitioning interrupts* correspond to the virtual hardware provided by the virtualization layer. It includes different virtual devices associated to the virtualization. Some of these virtual devices are:
 - Virtual global and local clocks and timers
 - New message arrival. The communication mechanism (channel) implemented by XtratuM is seen as a hardware device generating an interrupt when the operation is completed.
 - Partition slot execution. In a partitioned system the partition is aware of the partition scheduling, this interrupt informs to the partition that a new slot has been scheduled.

4.8.5.2 Interrupt model implementation

XtratuM provides a virtual vector for interrupt management at partition level. This VirtualTrapTable is a interrupt model that virtualizes the underlying interrupts available in the hardware and adds a set of new interrupts related to the partitioned system.

Figure 10 shows the scheme of a generic interrupt model.

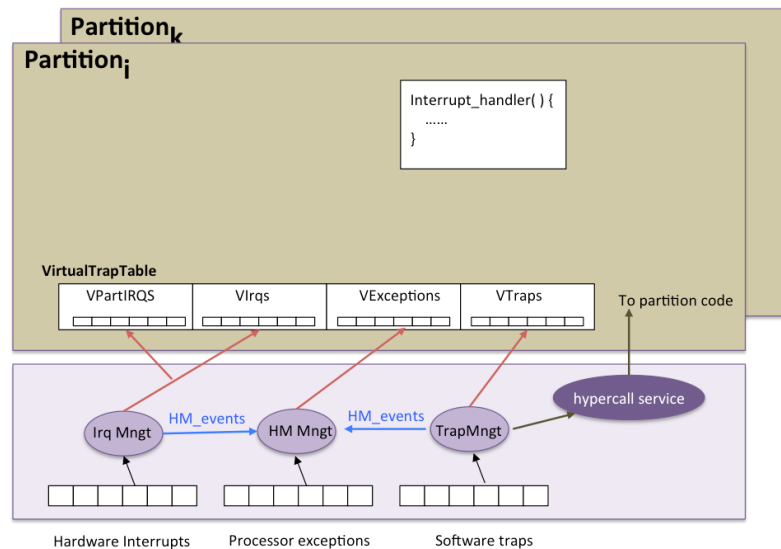


Figure 10: Interrupt Model

The API provides a set of symbols for virtual interrupts.

```
#define XM_VT_EXT_FIRST      (0)
#define XM_VT_EXT_LAST      (31)
#define XM_VT_EXT_MAX       (32)

// Virtual Exceptions
#define XM_HM_EV_ARM_UNDEF_INSTR (XM_HM_MAX_GENERIC_EVENTS+0)
#define XM_HM_EV_ARM_PREFETCH_ABORT (XM_HM_MAX_GENERIC_EVENTS+1)
#define XM_HM_EV_ARM_DATA_ABORT (XM_HM_MAX_GENERIC_EVENTS+2)
#define XM_HM_EV_ARM_DATA_ALIGNMENT_FAULT (XM_HM_MAX_GENERIC_EVENTS+3)
#define XM_HM_EV_ARM_DATA_BACKGROUND_FAULT (XM_HM_MAX_GENERIC_EVENTS+4)
#define XM_HM_EV_ARM_DATA_PERMISSION_FAULT (XM_HM_MAX_GENERIC_EVENTS+5)
#define XM_HM_EV_ARM_INSTR_ALIGNMENT_FAULT (XM_HM_MAX_GENERIC_EVENTS+6)
#define XM_HM_EV_ARM_INSTR_BACKGROUND_FAULT (XM_HM_MAX_GENERIC_EVENTS+7)
#define XM_HM_EV_ARM_INSTR_PERMISSION_FAULT (XM_HM_MAX_GENERIC_EVENTS+8)

// Virtual extended Partitioning IRQs
#define XM_VT_EXT_HW_TIMER      (0+XM_VT_EXT_FIRST)
#define XM_VT_EXT_EXEC_TIMER    (1+XM_VT_EXT_FIRST)
#define XM_VT_EXT_WATCHDOG_TIMER (2+XM_VT_EXT_FIRST)
#define XM_VT_EXT_SHUTDOWN      (3+XM_VT_EXT_FIRST)
#define XM_VT_EXT_SAMPLING_PORT (4+XM_VT_EXT_FIRST)
#define XM_VT_EXT_QUEUING_PORT  (5+XM_VT_EXT_FIRST)

#define XM_VT_EXT_CYCLIC_SLOT_START (8+XM_VT_EXT_FIRST)
```

4.8.6 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 try to solve or confine the faulting subsystem in order to avoid or reduce the possible consequences.

HM is executed as result of a HM event occurrence. Next scenarios can raise a HM event:

- An exception has been raised 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 validated.
- A trap has been received and the temporal or spatial properties are not validated.
- 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.

Previous cases cover all entry points to the hypervisor. As result of enforcing the isolation of the partitions, XtratuM performs a check of the temporal and spatial properties each time that it is invoked.

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.

XtratuM HM subsystem is composed by four logical components:

- HM configuration: to bind the occurrence of each HM event with the appropriate HM action. This bind is specified in the configuration file.
- HM event detection: to detect abnormal states, using logical assertions in the XtratuM code.
- HM actions: a set of predefined actions to recover the fault or confine the error.
- HM notification: to report the occurrence of the HM events.

Once a HM event is raised, XtratuM performs an action that is specified in the configuration file. Some of the HM events and HM actions are shown in the next table.

```
// HM EVENTS
#define XM_HM_EV_INTERNAL_ERROR 0
#define XM_HM_EV_UNEXPECTED_TRAP 1
#define XM_HM_EV_PARTITION_UNRECOVERABLE 2
#define XM_HM_EV_PARTITION_ERROR 3
#define XM_HM_EV_PARTITION_INTEGRITY 4
#define XM_HM_EV_MEM_PROTECTION 5
#define XM_HM_EV_OVERRUN 6
#define XM_HM_EV_SCHED_ERROR 7
```

```

#define XM_HM_EV_WATCHDOG_TIMER 8
#define XM_HM_EV_INCOMPATIBLE_INTERFACE 9
// HM ARCH EVENTS
#define XM_HM_EV_ARM_UNDEF_INSTR (XM_HM_MAX_GENERIC_EVENTS+0)
#define XM_HM_EV_ARM_PREFETCH_ABORT (XM_HM_MAX_GENERIC_EVENTS+1)
#define XM_HM_EV_ARM_DATA_ABORT (XM_HM_MAX_GENERIC_EVENTS+2)
#define XM_HM_EV_ARM_DATA_ALIGNMENT_FAULT (XM_HM_MAX_GENERIC_EVENTS+3)
#define XM_HM_EV_ARM_DATA_BACKGROUND_FAULT (XM_HM_MAX_GENERIC_EVENTS+4)
#define XM_HM_EV_ARM_DATA_PERMISSION_FAULT (XM_HM_MAX_GENERIC_EVENTS+5)
#define XM_HM_EV_ARM_INSTR_ALIGNMENT_FAULT (XM_HM_MAX_GENERIC_EVENTS+6)
#define XM_HM_EV_ARM_INSTR_BACKGROUND_FAULT (XM_HM_MAX_GENERIC_EVENTS+7)
#define XM_HM_EV_ARM_INSTR_PERMISSION_FAULT (XM_HM_MAX_GENERIC_EVENTS+8)
.....

//HM Actions
#define XM_HM_AC_IGNORE 0
#define XM_HM_AC_SHUTDOWN 1
#define XM_HM_AC_PARTITION_COLD_RESET 2
#define XM_HM_AC_PARTITION_WARM_RESET 3
#define XM_HM_AC_HYPERVISOR_COLD_RESET 4
#define XM_HM_AC_HYPERVISOR_WARM_RESET 5
#define XM_HM_AC_SUSPEND 6
#define XM_HM_AC_HALT 7
#define XM_HM_AC_PROPAGATE 8
#define XM_HM_AC_SWITCH_TO_MAINTENANCE 9

```

4.8.7 Partition image header

The partition image header is a data structure with the following fields:

```

struct xmImageHdr {
#define XMEF_PARTITION_MAGIC 0x24584d69 // $XMi
    xm_u32_t sSignature;                // start signature
    xm_u32_t compilationXmAbiVersion;  // XM's abi version
    xm_u32_t compilationXmApiVersion;  // XM's api version
    xm_u32_t noCustomFiles;            // Number of custom files
    struct xefCustomFile customFileTab[CONFIG_MAX_NO_CUSTOMFILES];
    xm_u32_t eSignature;                // end signature
} __PACKED;

```

where

- sSignature and eSignature: Holds the start and end signatures which identifies the structure as a XtratuM partition image.
- compilationXmAbiVersion: XtratuM ABI version used to compile the partition. That is, the ABI version of the libxm and other accompanying utilities used to build the XEF file.
- compilationXmApiVersion: XtratuM API version used to compile the partition. That is, the API version of the libxm and other accompanying utilities used to build the XEF file.

- noCustomFiles: The number of extra files accompanying the image. If the image were Linux, then one of the modules would be the `initrd` image. Up to `CONFIG_MAX_NO_FILES` can be attached.
- customFileTab: Table information about the customisation files.

The `xmImageHdr` structure has to be placed in a section named `".xmImageHdr"`. The remainder of the image is free to the partition developer. There is not a predefined format or structure of where the code and data sections shall be placed.

5 Booting process

The booting process describes how the system (hypervisor) is initialized. In most of the processors the boot process is initialized when the program counter register is initialized at a specific memory address. A small program “resident software” is in charge of the initial steps of booting the computer.

This resident software will be in charge of loading into memory the hypervisor, its configuration file (XM_CT) and the partitions. The information hold by the XM_CT file is used to load any partition image.

5.1 Hypervisor boot

When the control is transferred from the resident software to XtratuM, a setup() function starts the boot operation. It is sketched in Figure 11.

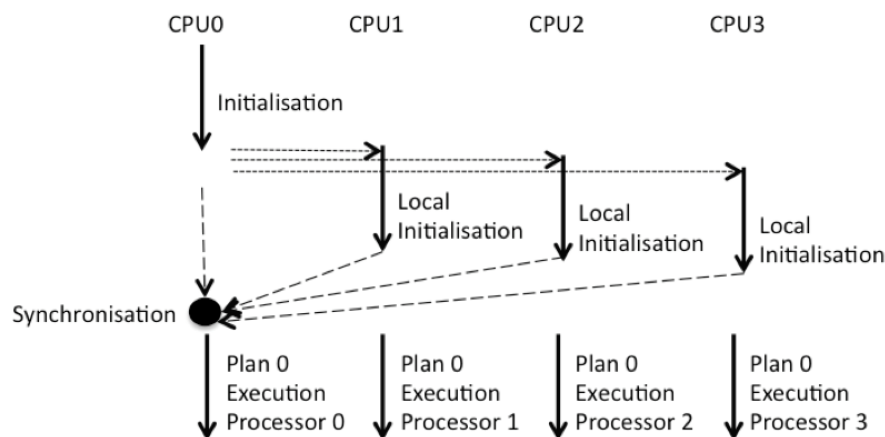


Figure 11: Booting a multicore architecture

After the hard reset, the CPU0 is started and a XtratuM thread is executed. This CPU0 thread performs a global initialisation and starts the execution of other CPUs by providing the entry point and stack area. Each started CPU executes a XtratuM thread performing a local initialisation of the internal local data structures. All CPU threads are synchronised in a specific point in order to guarantee a coherent initialisation before the scheduling plan execution.

The global initialisation consists on the following:

1. Initializes the internal console.
2. Initializes the interrupt controller.
3. Detects the processor frequency (information extracted from the XML configuration file).
4. Initializes memory manager (enabling XtratuM to keep track of the use of the physical memory).
5. Initializes hardware and virtual timers.
6. Initializes the scheduler.

7. Initializes the communication channels.
8. Wakes up other processors
9. Booting partitions are set in NORMAL state and non-booting ones are set in HALT state.
10. Opens the sync barrier
11. Finally, the setup function calls the scheduler and becomes into the idle task.

Other CPUs perform the local initialisation:

1. Sets up a valid virtual memory map
2. Initializes the timer
3. Waits in a sync barrier
4. Finally, calls the scheduler and becomes into the idle task.

This scheme implies an important design aspect with respect to the moncore version of XtratuM. The internal code of XtratuM is not a non preemptive code block like it is in the moncore version. The multicore design is fully preemptive. A set of low grain atomic sections have been defined in order to avoid race conditions between the internal threads of XtratuM.

5.2 Partition boot

XtratuM provides virtual CPUs to the partitions. A moncore partition will use the virtual CPU identified as vCPU0. Its operation is exactly the same as the moncore version of XtratuM.

After a partition reset, the vCPU0 is initialized to the default values specified in the configuration file. Although the moncore partition uses the vCPU0, it can be allocated to any of the available

A multicore partition can use several virtual CPUs (vCPU0, vCPU1, vCPU2, ...) to implement the partition. XtratuM follows the approach for virtual CPUs than the hardware provides. At partition boot, XtratuM only starts vCPU0 for the partition. It is responsibility of the partition code in the initialized vCPU0 thread to start the execution of the additional cores.

An important aspect to be considered is that the virtual CPUs are local to each partition. It means, each partition handles its virtual CPUs which are completely hidden to other partitions. In order to handle virtual CPUs, XtratuM provides some services (hypercalls) to partitions to handle its virtual CPUs.

6 System configuration

The system configuration is responsibility of the System Architect. He is in charge of the definition of the partitions, the resources and the information flows between partitions. This is specified in the configuration file (XM_CF in XtratuM notation) in XML format.

XtratuM enforces that each virtualized and exported resource can be accessed by a partition at a time. To achieve this goal, the security functions (XSF) ensure that partitions are executed according a cyclic plan specified in the configuration vector. This plan is analysed off-line to guarantee plan properties (i.e. no overlapped intervals have been specified).

For resources such as memory, which does not require mutual exclusion to the whole, the XSF provides full isolation by allocating physically distinct portions of the resource to different partitions. XSF ensures the spatial isolation of its internal resources. Subjects, and resources made available to subjects by the XSF, are identified as exported resources.

The Partitioned Information Flow Policy (PIFP) defines the rules for isolation granted by the virtualization layer. It defines the authorisations for information flow between partitions and between subjects and exported resources. It is generated from the XM_CF and allows to apply the internal security functions during the execution.

An information flow is defined as a $\langle \text{partition/subject}, \text{partition/exported resource}, \text{mode} \rangle$ triplet. Note that the exported resource may be another subject. All the information flows have to be specified in the XM_CF. By default, no information flow between partitions or between subjects and exported resources is allowed.

6.1 XtratuM subjects, objects and privileges

Based on the Common Criteria definitions [3,4,5],

- **Subjects** are active entities in the partitioned system that perform operations on objects. The subjects can be categorized in two types: privileged and normal.
- **Exported resources** are passive entities that contain or receive information, and upon which subjects perform operations.
- **Operation mode** (on a resource) is a specific type of action performed by a subject on an exported resource.

6.1.1 Subject identification

XtratuM manages partitions as its main active entities. Processes inside of a partition are handled internally and XtratuM does not know of their existence. Partitions are responsible of the internal process management. Any operation performed by any of the internal active elements of a partition is seen as a partition operation.

Based on this approach, the set S of subjects is formed by all the partitions defined in the XM_CF.

$$S = S_0, S_1, S_2, \dots, S_n$$

6.1.2 Exported resource identification

XtratuM gets the system information via the XM_CF. In this configuration vector, all the subjects, exported resources and operations have to be defined beforehand.

XtratuM provides a para-virtualization of some exported resources whereas others are accessed "directly" by the partition. This "directly" means that the information flow has been explicitly authorised by the Virtualization Layer (VLayer).

Exported resources can be classified as:

- **Processor and register:** Exported resources that are exported as a whole during a temporal window.
 - **VCPU:** Virtual CPUS. It includes the internal registers (user and control registers). The scheduling plan identifies *which* subject will use this resource and *when*. (ER^{VCPU} , ER^{URg} , ER^{CRg}).
 - **VFPU:** Virtual Floating Point Units. It is exported jointly with the VCPU if the subject has specified its use in the XM_CF. (ER^{VFPU})
- **Time management and IRQs:** Clock, timers and interrupts are virtualized to the subjects. No direct access from the subject is allowed. XtratuM provides a mechanism via hypercalls to access indirectly to these resources. (ER^{Tim} , ER^{IRQ}).
- **Memory areas:** Memory is not exported as a whole. Memory areas are regions of memory that are directly exported as resource to subjects. Each subject is allowed direct access to specific memory areas defined in the XM_CF. (ER^{MAk})
 - **Memory layout:** defines the whole memory available in the system. All memory areas exported have to be independent (no overlapped) subsets of the memory layout. It is not directly defined as an exportable resource, it is exported as a resource via memory area definitions.
 - **Memory area:** defines a memory region as an exported resource. It includes the memory needed by the subjects to be executed, the shared memory between subjects and memory block devices.
 - **IO memory area:** defines a IO memory region as exported resource. These exported resources are assigned exclusively to a subject (no shareable).
- **Basic peripherals:** These devices (UART,) are basic components of the system that are exported as resources. They require an explicit definition and subjects using this resource have to explicitly declare it. The access to these devices is done via hypercalls. (ER^{UART})
- **Inter-partition communications:** XtratuM provides an inter-partition communication mechanism based on *channels*. It allows to subjects to send/receive messages to/from channels using *ports*. (ER^{CHNk}).
 - **Channels:** They have to be specified in the XM_CF. They are not directly exported resources. Channels are seen by subjects through *ports*. Channels are specified in the XMCF in order to link information flow sources (*<subject, port, SOURCE>*) with information flow destination (*<subject, port, DESTINATION>*).
 - **Ports:** are exported resources that have to be specified in the XMCF. They are accessed via hypercalls.

- **Time allocation:** specifies when the exported resources are available to subjects. It is defined by means of temporal windows or slots in the scheduling plan and scheduling modes. (ER^{PLNk} , ER^{SLTk}).
 - **Plan:** is an exportable resource that defines an execution cyclic scheme for the subjects.
 - **Slot:** is an exportable resource that specifies the time interval allocated to a specific subject.
- **Traces:** are exported resources that permit authorized subjects to register events (traces/audit records) (ER^{TRk})

6.1.3 Exported resource access mechanism

The following table summarizes the mechanisms used by subjects to access to the exported resources as well as where the control is done, the mechanism used by the PIFP.

ER	Control Place	Mechanism	PIFP	Comments
VCPU time	Hw/Core	direct/hypercall	explicit	Specified in the plan
VCPU User Registers	Hw/Core	direct/hypercall	implicit	Specified in the plan
VCPU Ctrl Registers	Hw/Core	hypercall	implicit	All
VFPU	Hw/Core	direct	explicit	flags in the partition definition
Memory Areas	Hw	direct	explicit	Memory areas definition in the partition
MMU	Hw	direct	implicit	
Traps	Hw	hypercall	parameterised	IRQ lines in the partition
Timers	Core	Hypercalls	implicit	
I/O resources	Core	Hw/hypercall	explicit	IO devices

For logical resources:

ER	Control Place	Mechanism	PIFP	Comments
System modes	Core	hypercall	Explicit	plan definition
Partition states	Core	hypercall	Implicit	
IPC	Core	hypercalls	Explicit	Channel definition and port in partitions
HM	Hardware/Core	hypercall	Explicit	HM action in partitions (default values)
Partition Traces	Core	hypercall	Explicit	Trace definition in partition

6.1.4 Operations on exported resources

The identified operations on exported resources are:

- **Read.** A subject can read the exported resource.
- **Write.** A subject can write on the exported resource.
- **Run.** A subject it is only allowed to use the exported resources during the run operation.

6.1.5 Partitions and the Partitioned Information Flow Policy (PIFP)

The virtualization layer provides partitions as abstraction implemented by the XSF. XtratuM manages partitions. The guest OS personality is in charge of managing the internal subjects (threads or tasks or processes) that are not visible from XtratuM.

From this point of view, it is assumed the **Partition Abstraction policy**: *The subjects in a partition have homogeneous requirements for access, on a per-partition basis, to exported resources.*

It is responsibility of the guest OS to define another policy. For instance, a guest OS could define a **Least Privilege Abstraction** which assumes that *the subjects in a partition have heterogeneous requirements for access to exported resources*. In this case, the guest OS could restrict the operations defined in the other policy to some internal subjects.

6.1.6 Access matrices

The access matrix specifies the privileges that each subject has over each object. To define the subjects type: privilege and normal.

Table access matrix is generated from the configuration vector and, consequently, from the configuration file XM-CF.

Next listing shows an example of XM_CF for LEON3 processor.

```
<SystemDescription xmlns="http://www.xtratum.org/xm-3.x"
  version="1.0.0" name="hello_world">
  <HwDescription>
    <ProcessorTable>
      <Processor id="0" frequency="50Mhz">
        <CyclicPlanTable>
          <Plan id = "0" majorFrame="25ms">
            <Slot id="0" start="0ms" duration="10ms" partitionId="0"/>
            <Slot id="1" start="15ms" duration="5ms" partitionId="1"/>
          </Plan>
          <Plan id = "1" majorFrame="10ms">
            <Slot id="0" start="0ms" duration="5ms" partitionId="0"/>
            <Slot id="1" start="5ms" duration="5ms" partitionId="2"/>
          </Plan>
        </CyclicPlanTable>
      </Processor>
    </ProcessorTable>

    <MemoryLayout>
      <Region type="stram" start="0x40000000" size="4MB"/>
      <Region type="sdram" start="0x60000000" size="16MB"/>
    </MemoryLayout>
  </HwDescription>
</SystemDescription>
```

```

    </MemoryLayout>
  </HwDescription>

  <XMHypervisor console="Uart">
    <PhysicalMemoryAreas>
      <Area start="0x40000000" size="512KB" />
    </PhysicalMemoryAreas>
  </XMHypervisor>

  <PartitionTable>
    <Partition id="0" name="Partition1" flags="system" console="Uart">
      <PhysicalMemoryAreas>
        <Area start="0x40100000" size="256KB" />
        <Area start="0x40300000" size="128KB" flags="shared" />
      </PhysicalMemoryAreas>
      <TemporalRequirements duration="25ms" period="10ms"/>
      <PortTable>
        <Port name="writerQ" type="queuing" direction="source" />
        <Port name="writerS" type="sampling" direction="source" />
      </PortTable>

      <Trace device="Trace1"/>
      <HealthMonitor>
        <Event action="XM_HM_AC_HALT" log="yes"
          name="XM_HM_EV_PARTITION_ERROR" />
      </HealthMonitor>
      <HwResources>
        <Interrupts lines="4" />
      </HwResources>
    </Partition>

    <Partition id="1" name="Partition2" flags="fpu" console="Uart">
      <PhysicalMemoryAreas>
        <Area start="0x40180000" size="256KB" />
        <Area start="0x40300000" size="128KB" flags="shared" />
      </PhysicalMemoryAreas>
      <TemporalRequirements duration="25ms" period="5ms"/>
      <PortTable>
        <Port name="readerQ" type="queuing" direction="destination" />
        <Port name="readerS" type="sampling" direction="destination" />
      </PortTable>
      <Trace device="Trace2"/>
      <HwResources>
        <IoPorts>
          <Range base="0x80000080" noPorts="4"/>
          <Range base="0x80100110" noPorts="15"/>
          <Restricted address="0x80100200" mask="0x60"/>
        </IoPorts>
        <Interrupts lines="7"/>
      </HwResources>
    </Partition>

    <Partition id="2" name="Partition3" console="Uart">
      <PhysicalMemoryAreas>
        <Area start="0x40200000" size="256KB" />
      </PhysicalMemoryAreas>
      <TemporalRequirements duration="25ms" period="5ms"/>
      <PortTable>
        <Port name="readerS" type="sampling" direction="destination" />
      </PortTable>

```

```

    </Partition>
  </PartitionTable>

  <Channels>
    <QueuingChannel maxMessageLength="512B" maxNoMessages="10">
      <Source partitionId="0" portName="writerQ" />
      <Destination partitionId="1" portName="readerQ" />
    </QueuingChannel>
    <SamplingChannel maxMessageLength="512B">
      <Source partitionId="0" portName="writerS" />
      <Destination partitionId="2" portName="readerS" />
    </SamplingChannel>
  </Channels>

  <Devices>
    <MemoryBlock name="SystemTrace" start="0x40380000" size="128KB"/>
    <MemoryBlock name="Trace1" start="0x403C0000" size="64KB"/>
    <MemoryBlock name="Trace2" start="0x403E0000" size="64KB"/>
    <Uart id="0" baudRate="115200" name="Uart" />
  </Devices>
</SystemDescription>

```

The following elements are identified:

- Subjects
 - P1: Partition1 (System)
 - P2: Partition2
 - P3: Partition3
- Exported resources:
 - ER^{CPU}: VCPU time
 - ER^{URg}: CPU U. reg.
 - ER^{CRg}: CPU Ctl. reg.
 - ER^{FPU}: FPU
 - ER^{MMU}: MMU
 - ER^{TRP}: Traps
 - ER^{Tim}: Timers
 - ER^{MA1}: Memory layout region start="0x40000000" size="4MB"
 - ER^{MA2}: Memory layout region start="0x60000000" size="16MB"
 - ER^{MA3}: Memory area start="0x40100000" size="256KB"
 - ER^{MA4}: Memory area start="0x40180000" size="256KB"
 - ER^{MA5}: Memory area start="0x40200000" size="256KB"
 - ER^{MA6}: Shared Memory area start="0x40300000" size="256KB"
 - ER^{MA7}: Memory block area name="SystemTrace" start="0x40380000" size="128KB"
 - ER^{MA8}: Memory block area name="Trace1" start="0x403C0000" size="64KB"/>
 - ER^{MA9}: Memory block area name="Trace2" start="0x403E0000" size="64KB"/>
 - ER^{PLn1}: Plan id="0" majorFrame="25ms"
 - ER^{PLn2}: Plan id="1" majorFrame="10ms"
 - ER^{SLt1}: Slot id="0" start="0ms" duration="10ms" partitionId="0" plan id="0"
 - ER^{SLt2}: Slot id="1" start="15ms" duration="5ms" partitionId="1" plan id="0"
 - ER^{SLt3}: Slot id="0" start="0ms" duration="5ms" partitionId="0" plan id="1"
 - ER^{SLt4}: Slot id="1" start="5ms" duration="5ms" partitionId="1" plan id="1"
 - ER^{CHN1}: QueuingChannel maxMessageLength="512B" maxNoMessages="10" [<"0", "writerQ">,<"1", "readerQ">]
 - ER^{HN2}: SamplingChannel maxMessageLength="512B" [<"0", "writerS">,<"2", "readerS">]

- ER^{UART} : Uart id="0" baudRate="115200" name="Uart"
- ER^{IOM1} : IoPort base="0x80000080" noPorts="4"
- ER^{IOM2} : IoPort base="0x80100110" noPorts="15"
- ER^{IOM3} : IoPort Restricted address="0x80100200" mask="0x60"
- ER^{IRQ1} : Interrupts lines="4"
- ER^{IRQ2} : Interrupts lines="7"

6.1.7 Subject temporal allocation

For each virtual CPU the following table shows the partition allocation to temporal windows

Plans	Slots	S1	S2	S3
ER^{PLn1}	ER^{SLt1}	run		
	ER^{SLt2}		run	
ER^{PLn2}	ER^{SLt3}	run		
	ER^{SLt4}			run

6.1.8 Subject memory areas allocation

The virtualization layer manages the memory layout and makes areas of it accessible to subjects according to the configuration file. The following table shows the allocation of physical memory from the configuration file example.

ER	S1	S2	S3	Description
ER^{MA1}				start:"0x40000000" size:"4MB"
ER^{MA2}				start:"0x60000000" size:"16MB"
ER^{MA3}	rw			start:"0x40100000" size:"256KB"
ER^{MA4}	rw	rw		start:"0x40180000" size:"256KB"
ER^{MA5}	rw		rw	start:"0x40200000" size:"256KB"
ER^{MA6}	rw	rw		start:"0x40300000" size:"256KB"
ER^{MA7}	rw			start:"0x40380000" size:"128KB"
ER^{MA8}	rw			start:"0x403C0000" size:"64KB"
ER^{MA9}	rw			start:"0x403E0000" size:"64KB"
ER^{IOM1}		rw		base="0x80000080" noPorts="4"
ER^{IOM2}		rw		base="0x80100110" noPorts="15"
ER^{IOM3}		rw		address="0x80100200" mask="0x60"

6.1.9 Subjects and virtualized exported resources

ER	S1	S2	S3	Description
ER^{VCPU}	rw	rw	rw	VCPU time
ER^{URg}	rw	rw	rw	VCPU User registers
ER^{CRg}	rw	rw	rw	VCPU control registers

ER ^{VFP}	rw			VFP
ER ^{MMU}	rw	rw	rw	MMU
ER ^{TRP}	rw	rw	rw	Traps
ER ^{Tim}	rw	rw	rw	Timers

6.1.10 IPC exported resources

The IPC allocation to partitions is detailed in the next table.

ER	S1	S2	S3	Description
ER ^{CHN1}	w	r		QueuingChannel S1 => S2
ER ^{CHN2}	w		r	SamplingChannel S1 => S3

6.1.11 Devices exported resources

The Device allocation to partitions is detailed in the next table.

ER	S1	S2	S3	Description
ER ^{UART}	w	w	w	UART
ER ^{IRQ1}	w			lines="4"
ER ^{IRQ2}		w		lines="7"

6.2 Configuration file specification

All the components involved in the previous model are detailed in the configuration file. The configuration file follows a XML syntax specified by an XMLSchema model. The complete XML Schema configuration specification for ARM and X86 processors are included in the Appendix 1 of this document.

The configuration model includes the main elements:

- **Hardware description:** describes the underlying hardware
- **XMHypervisor:** describes the allocation of XtratuM
- **ResidentSw:** details the allocation of the resident software
- **PartitionTable:** includes the partitions in the system
- **Channels:** describes the communication channels

Figure 12 shows a graphical view of the configuration file elements. Figure 13 shows the description of the root element "SystemDescription" and the following subsections depict the main elements of the configuration specification.

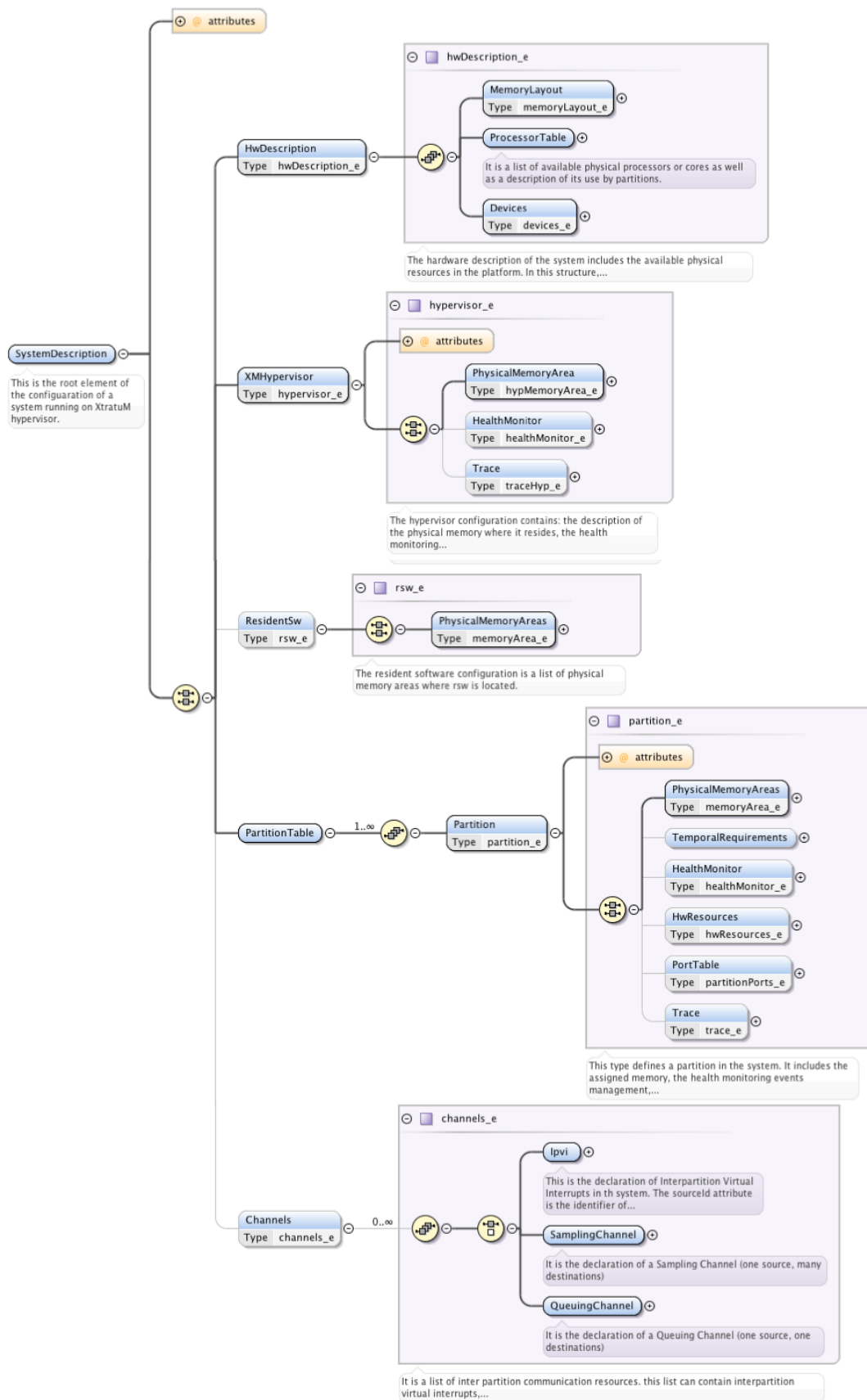


Figure 12: Graphical view of an example XM CF configuration file

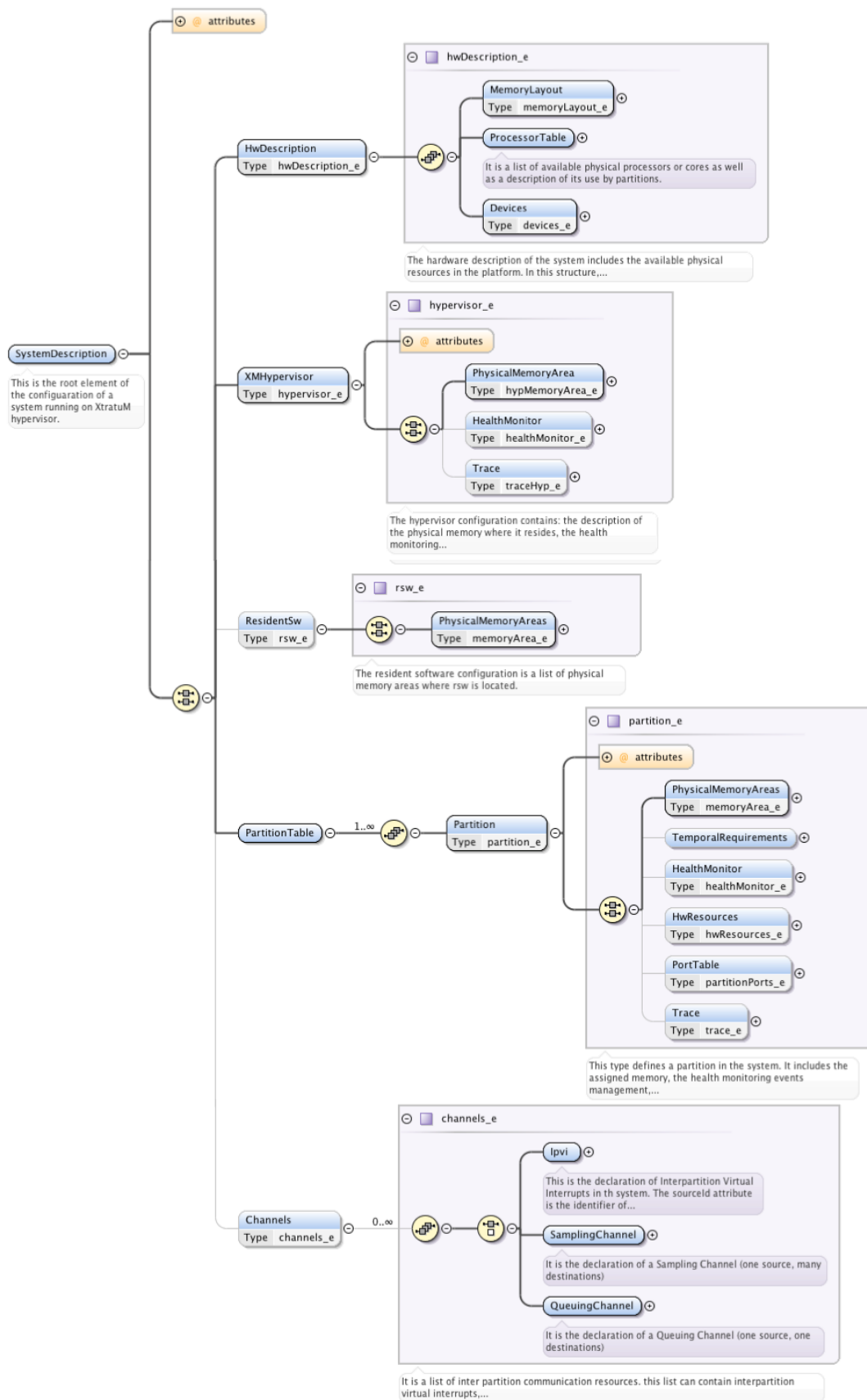


Figure 13: System Description Schema.

6.2.1 Element HwDescription

It describes the underlying hardware.

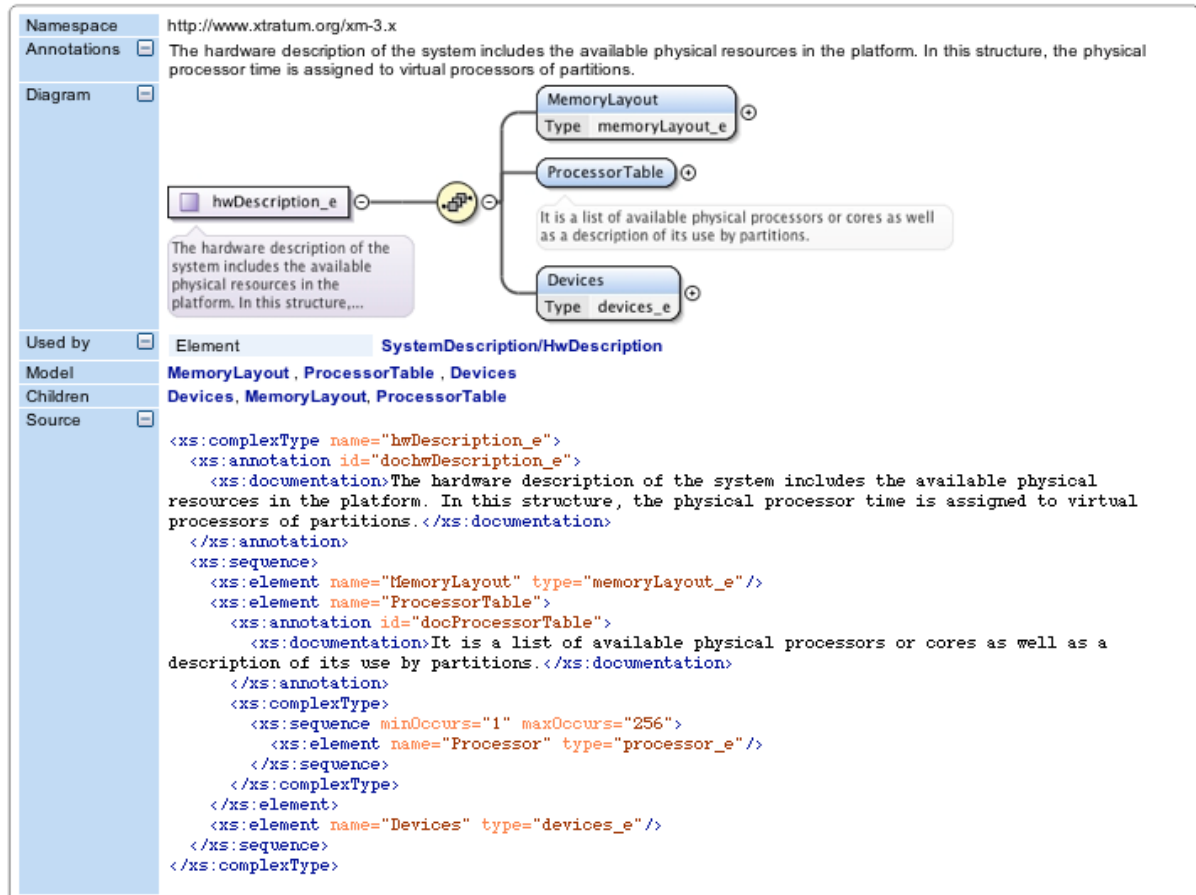


Figure 14: Hardware description component.

6.2.1.1 Element HwDescription/MemoryLayout



Figure 15: Hardware memory layout.

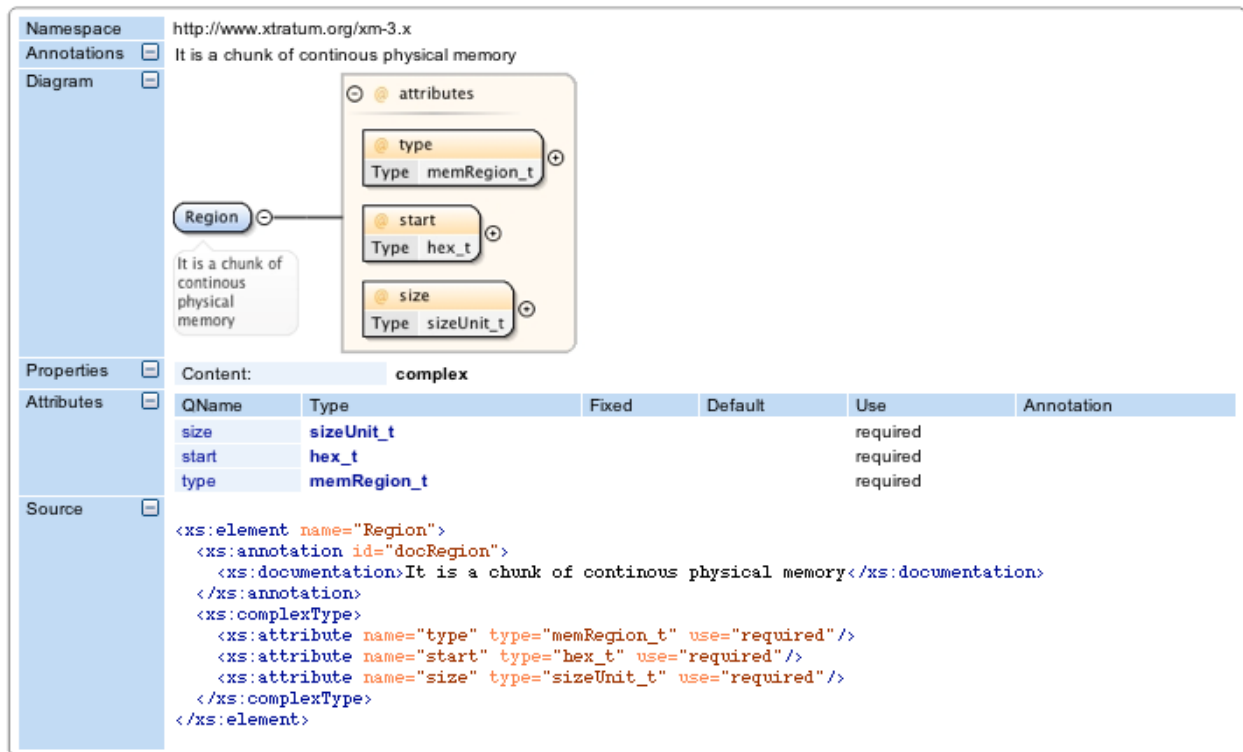


Figure 16: Hardware memory layout region.

6.2.1.2 Element HwDescription/ProcessorTable

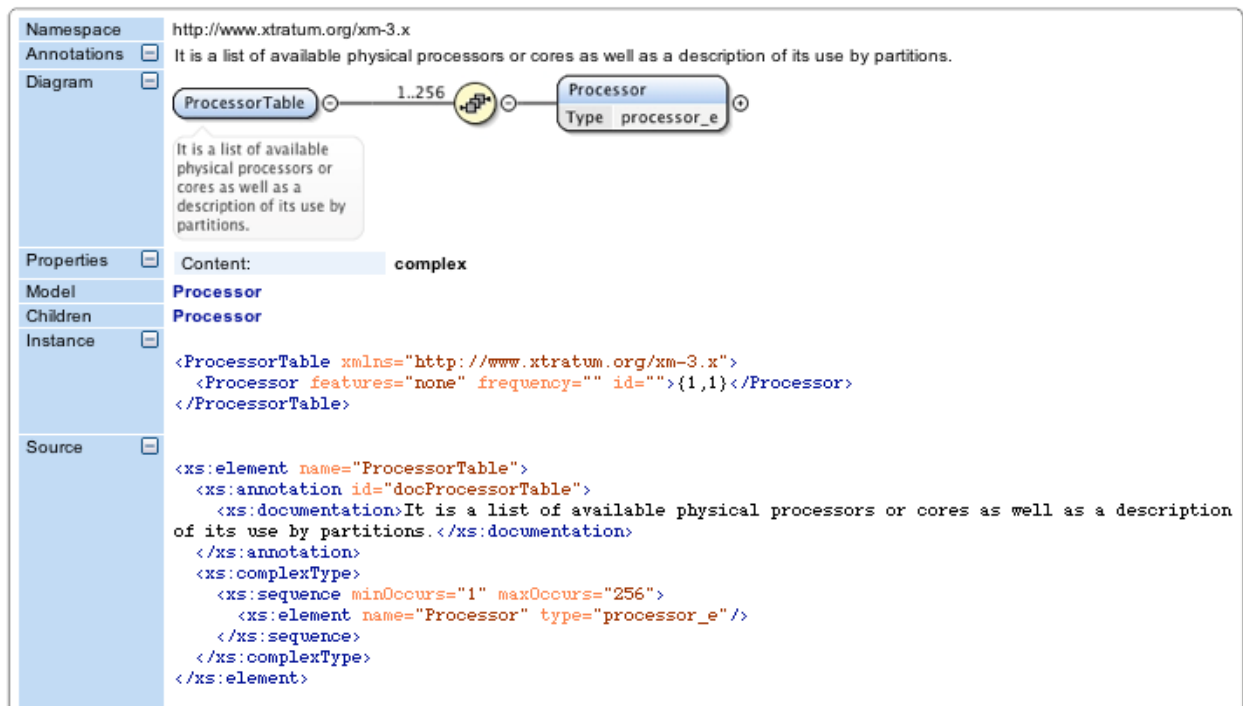


Figure 17: Processor table.

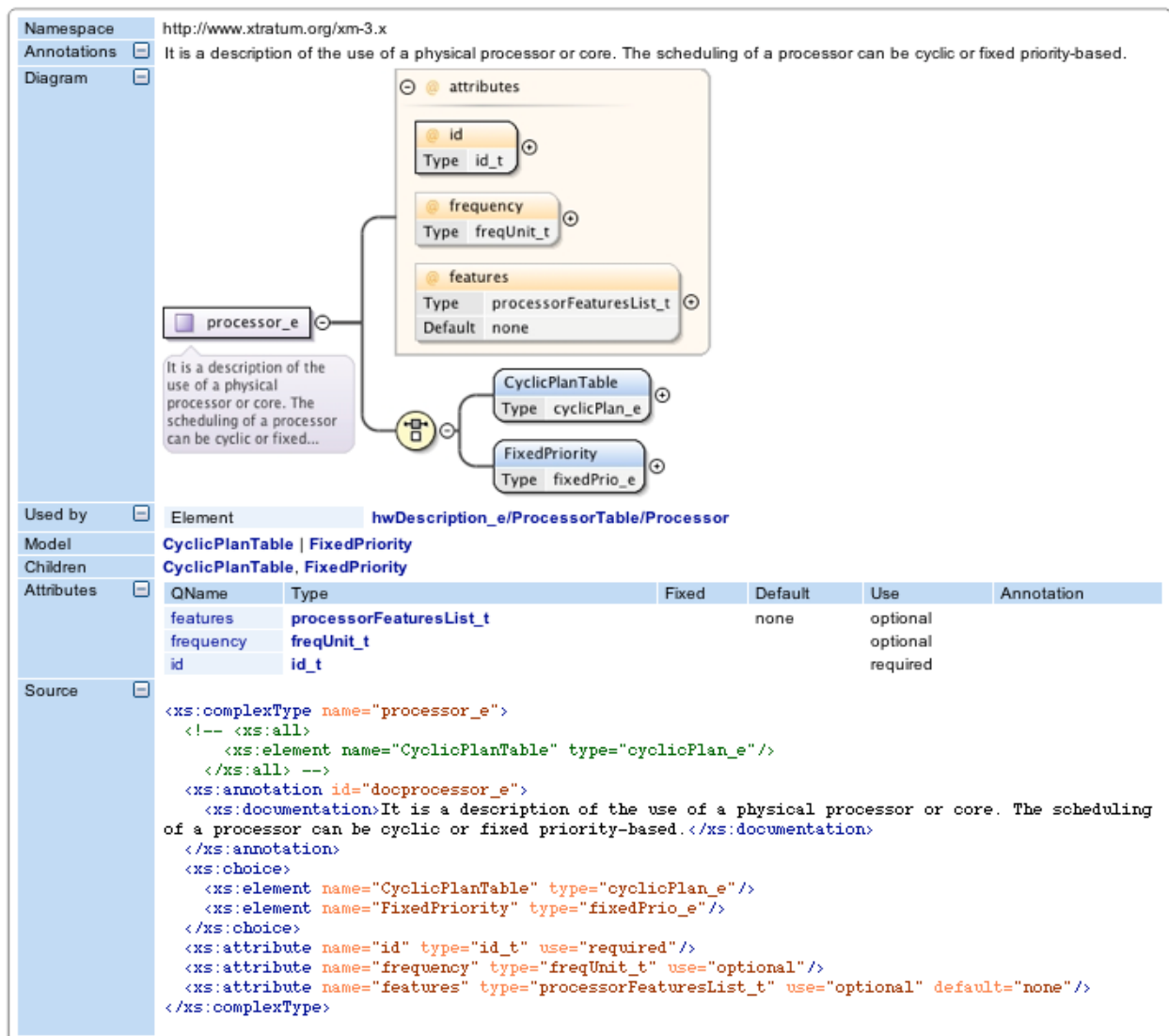


Figure 18: Processor.

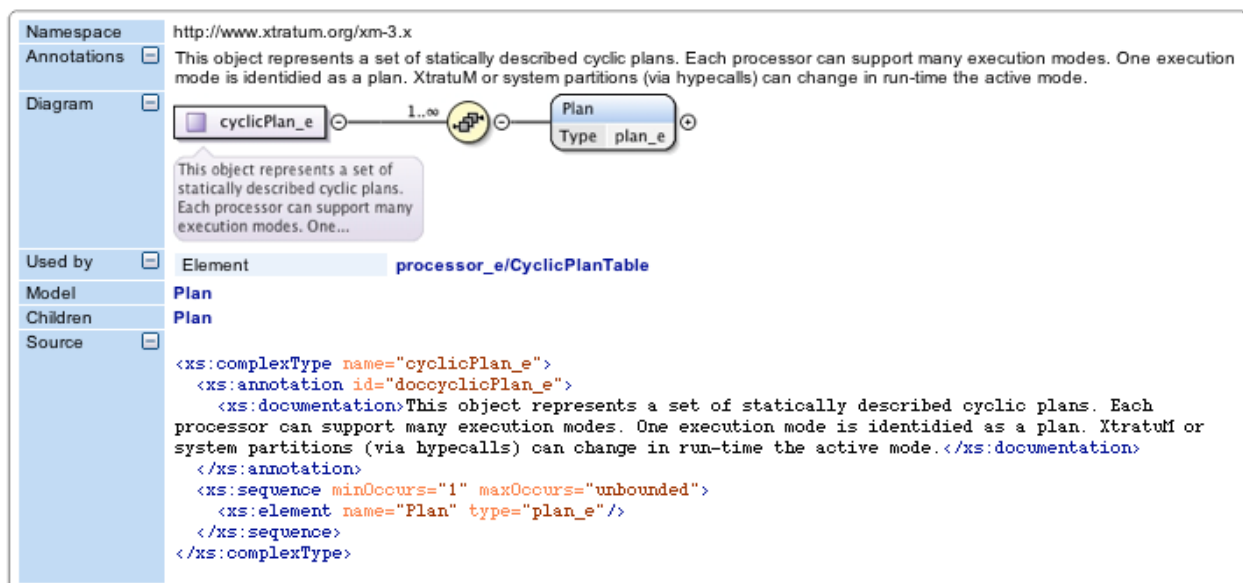


Figure 19: Cyclic plan scheduling.

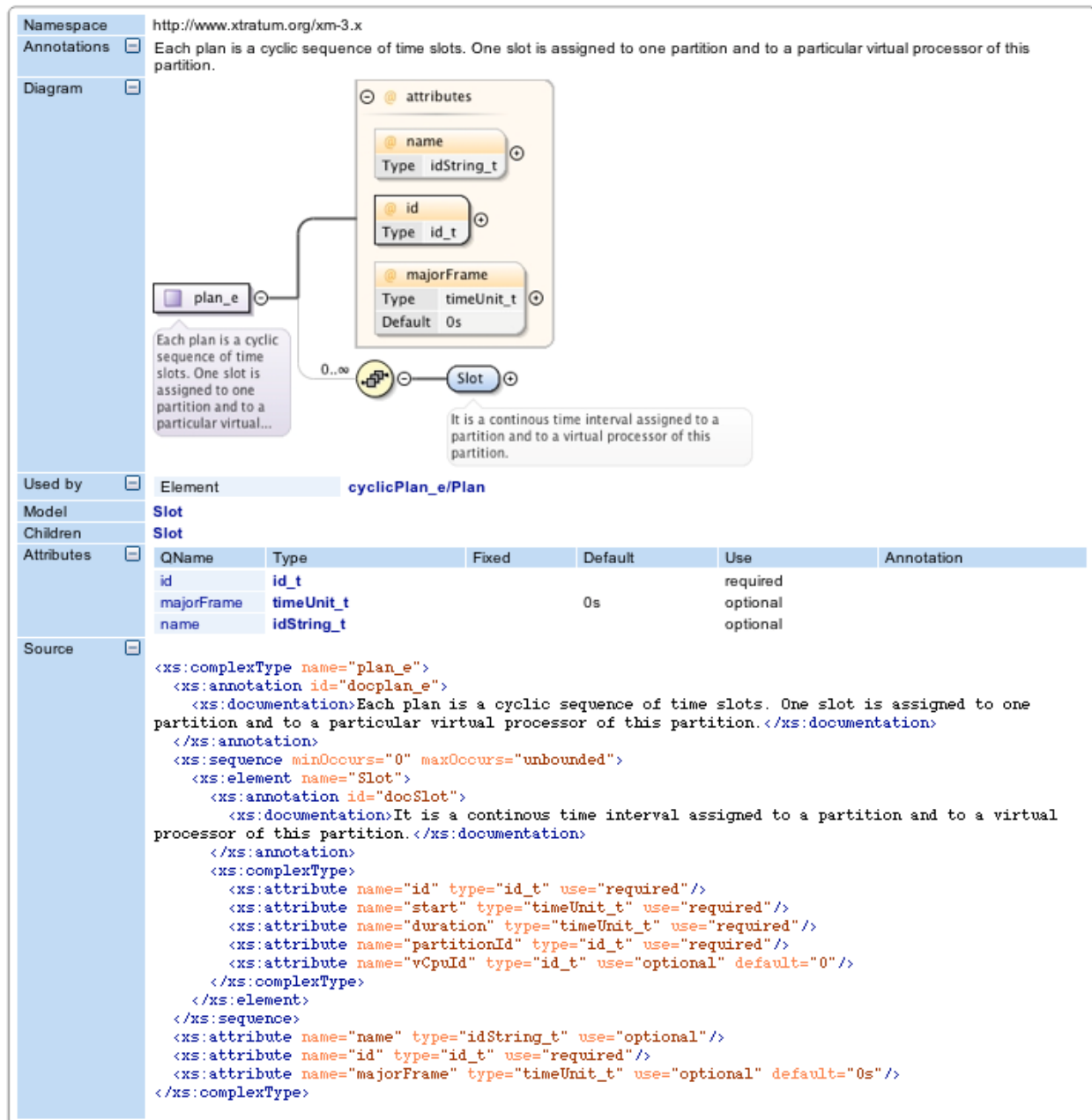


Figure 20: Plan of a cyclic plan scheduling.

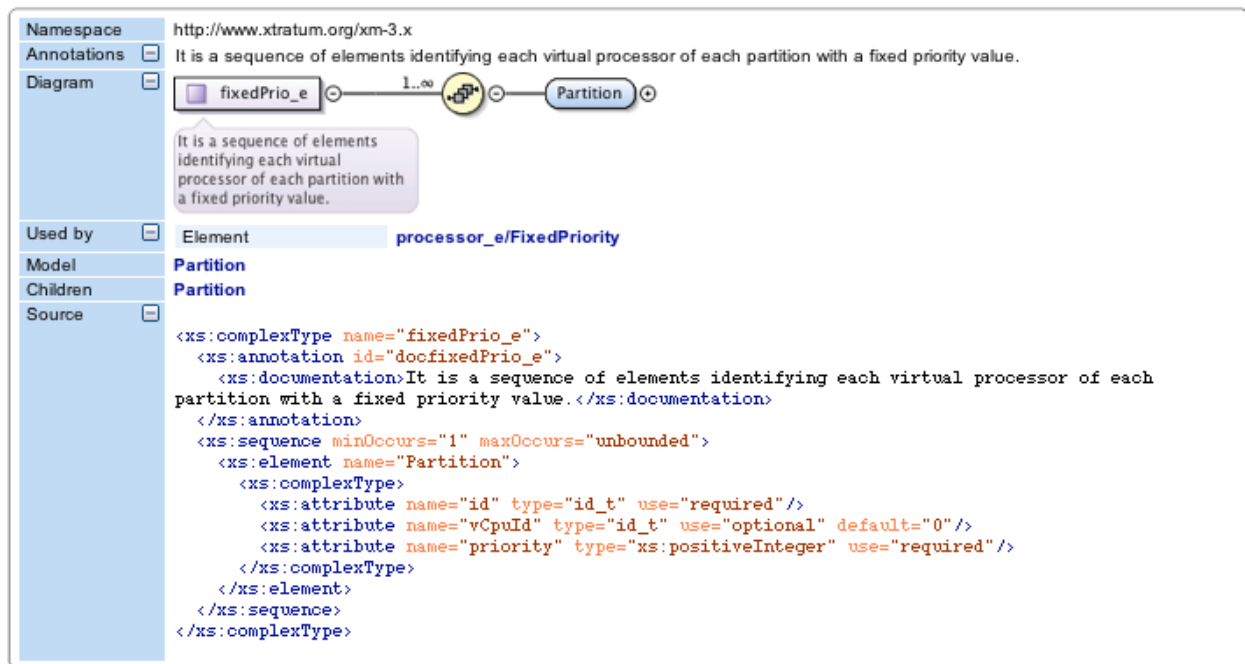


Figure 21: Fixed priority scheduling.

6.2.1.3 Element *HwDescription/Devices*

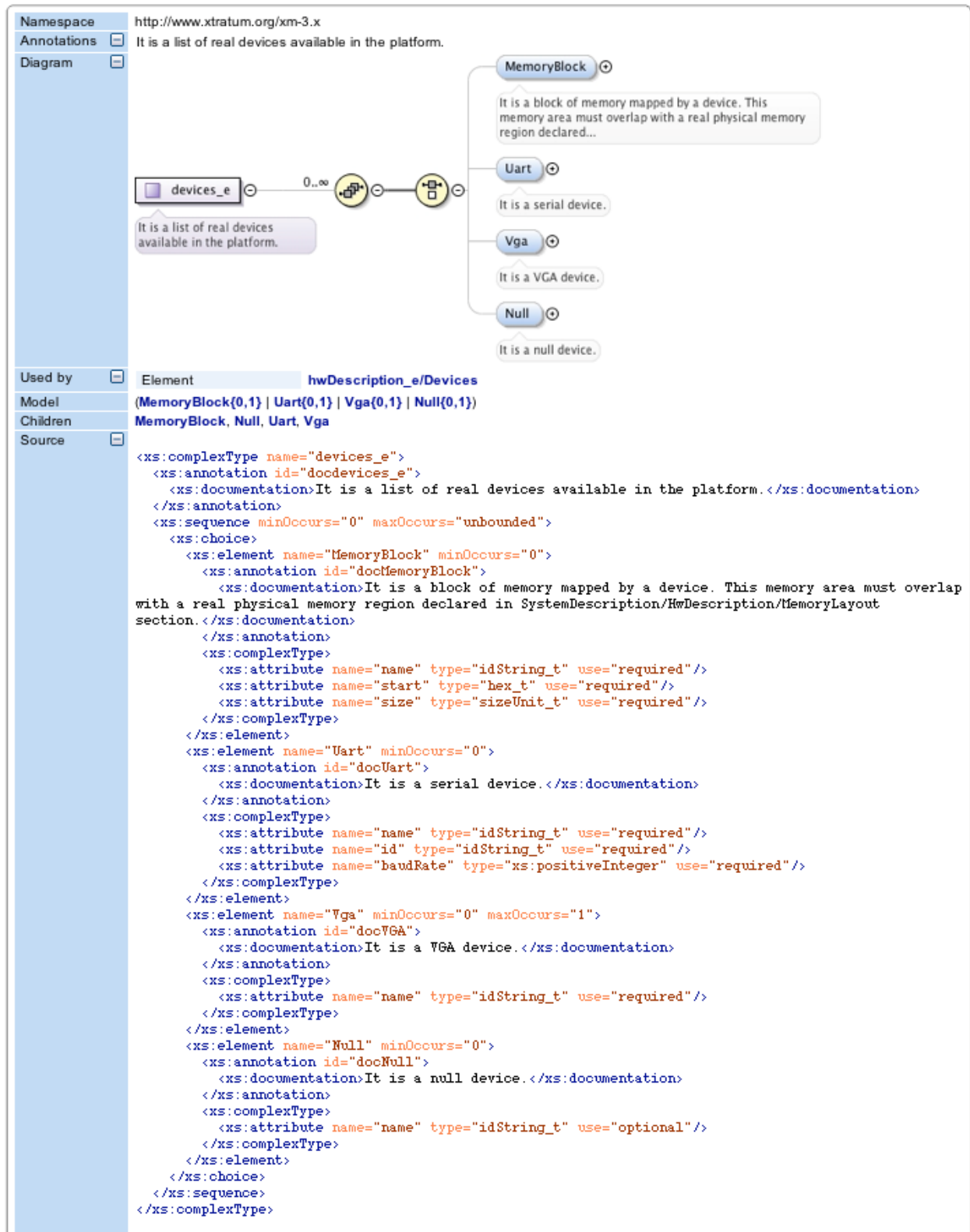


Figure 22: Hardware devices.

6.2.2 Element XMHypervisor

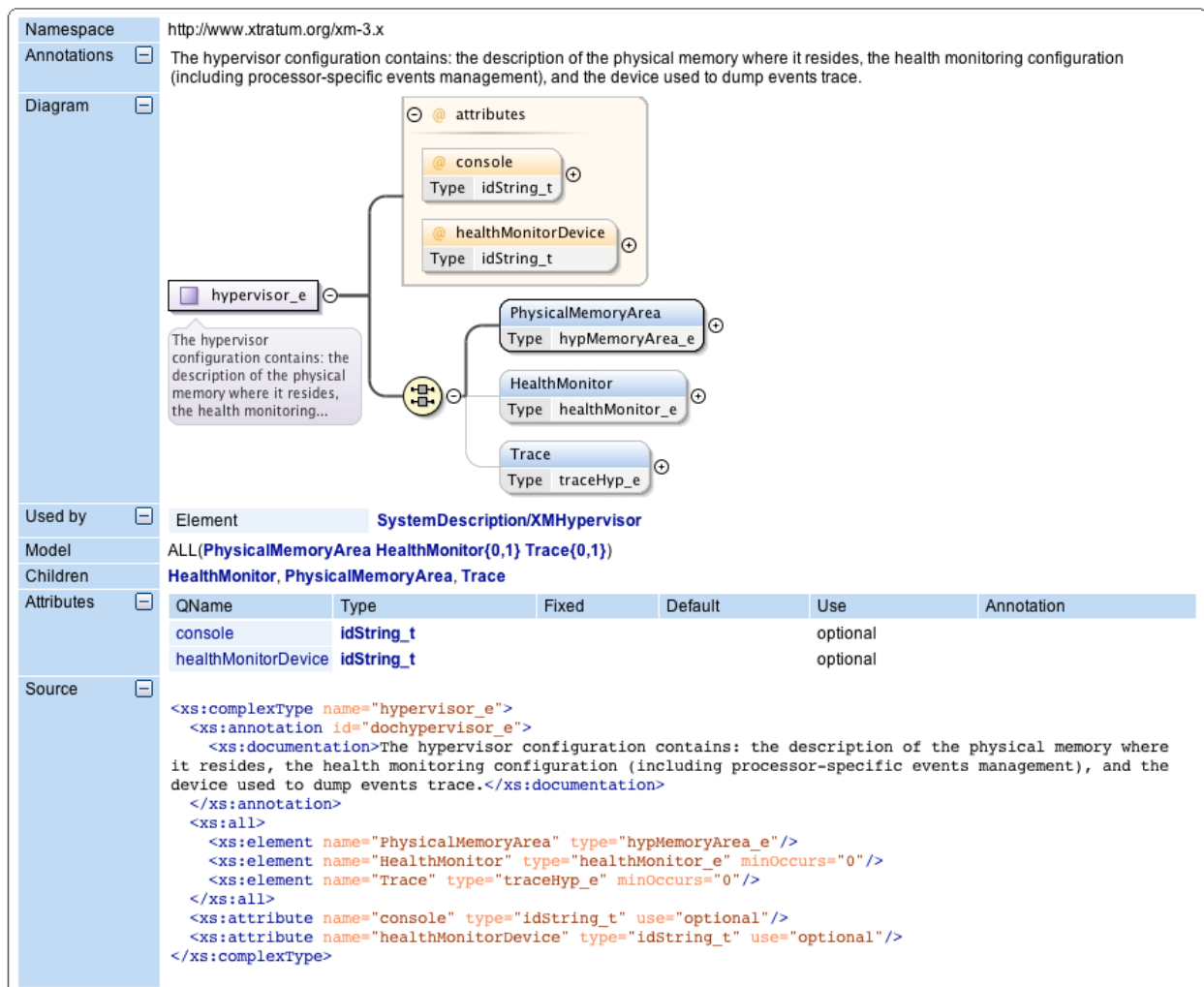


Figure 23: Hypervisor component.

6.2.2.1 Element *XMHypervisor/PhysicalMemoryArea*

Namespace	http://www.xratum.org/xm-3.x																							
Annotations	<input type="checkbox"/> This structure describes the continuous memory area that the hypervisor requires to be loaded. Size and access flags are defined but the load memory address is not yet defined at this level. The load address is part of the source code configuration and therefore hardcoded. The default value is 0x40000000.																							
Diagram	<input type="checkbox"/>																							
Used by	<input type="checkbox"/> Element hypervisor_e/PhysicalMemoryArea																							
Attributes	<table border="1"> <thead> <tr> <th>QName</th> <th>Type</th> <th>Fixed</th> <th>Default</th> <th>Use</th> <th>Annotation</th> </tr> </thead> <tbody> <tr> <td>flags</td> <td>memAreaFlagsList_t</td> <td></td> <td></td> <td>optional</td> <td></td> </tr> <tr> <td>size</td> <td>sizeUnit_t</td> <td></td> <td></td> <td>required</td> <td></td> </tr> </tbody> </table>						QName	Type	Fixed	Default	Use	Annotation	flags	memAreaFlagsList_t			optional		size	sizeUnit_t			required	
QName	Type	Fixed	Default	Use	Annotation																			
flags	memAreaFlagsList_t			optional																				
size	sizeUnit_t			required																				
Source	<input type="checkbox"/> <pre> <xs:complexType name="hypMemoryArea_e"> <xs:annotation id="dochypMemoryArea_e"> <xs:documentation>This structure describes the continuous memory area that the hypervisor requires to be loaded. Size and access flags are defined but the load memory address is not yet defined at this level. The load address is part of the source code configuration and therefore hardcoded. The default value is 0x40000000.</xs:documentation> </xs:annotation> <xs:attribute name="size" type="sizeUnit_t" use="required"/> <xs:attribute name="flags" type="memAreaFlagsList_t" use="optional"/> </xs:complexType> </pre>																							

Figure 24: Hypervisor memory.

6.2.2.2 Element *XMHypervisor/HealthMonitor*

Namespace	http://www.xratum.org/xm-3.x					
Annotations	<input type="checkbox"/> This structure configures the health monitoring of the hypervisor. It is a list of events that must be managed by the health monitor at hypervisor level. Events are architecture-specific. Automatic actions can be defined for each managed event as well as a boolean value indicating if the event must be logged into the log device of the hypervisor.					
Diagram	<input type="checkbox"/>					
Used by	<input type="checkbox"/> Elements hypervisor_e/HealthMonitor, partition_e/HealthMonitor					
Model	Event					
Children	Event					
Source	<input type="checkbox"/> <pre> <xs:complexType name="healthMonitor_e"> <xs:annotation id="dochealthMonitor_e"> <xs:documentation>This structure configures the health monitoring of the hypervisor. It is a list of events that must be managed by the health monitor at hypervisor level. Events are architecture-specific. Automatic actions can be defined for each managed event as well as a boolean value indicating if the event must be logged into the log device of the hypervisor.</xs:documentation> </xs:annotation> <xs:sequence minOccurs="1" maxOccurs="unbounded"> <xs:element name="Event"> <xs:annotation id="docEvent"> <xs:documentation>Each Event element defines a relationship between a health monitoring event and the actions that must be performed when it is raised.</xs:documentation> </xs:annotation> <xs:complexType> <xs:attribute name="name" type="hmString_t" use="required"/> <xs:attribute name="action" type="hmAction_t" use="required"/> <xs:attribute name="log" type="yntf_t" use="required"/> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </pre>					

Figure 25: Hypervisor health monitor.

6.2.2.3 Element *XM*Hypervisor/Trace

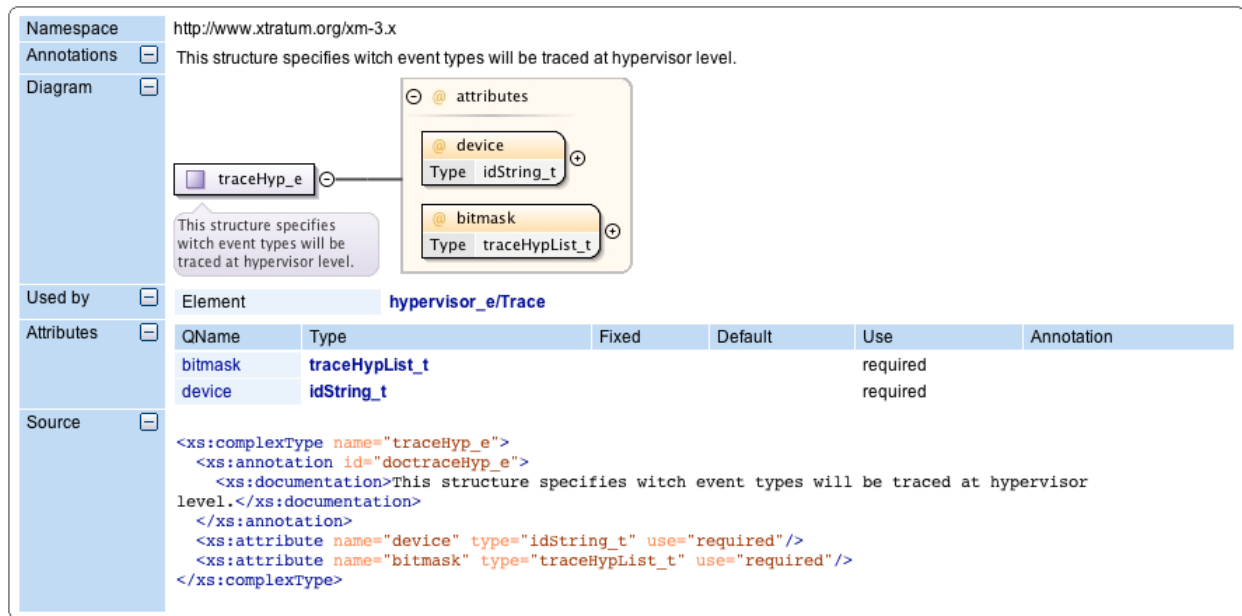


Figure 26: Hyperviosr trace.

6.2.3 Element ResidentSw

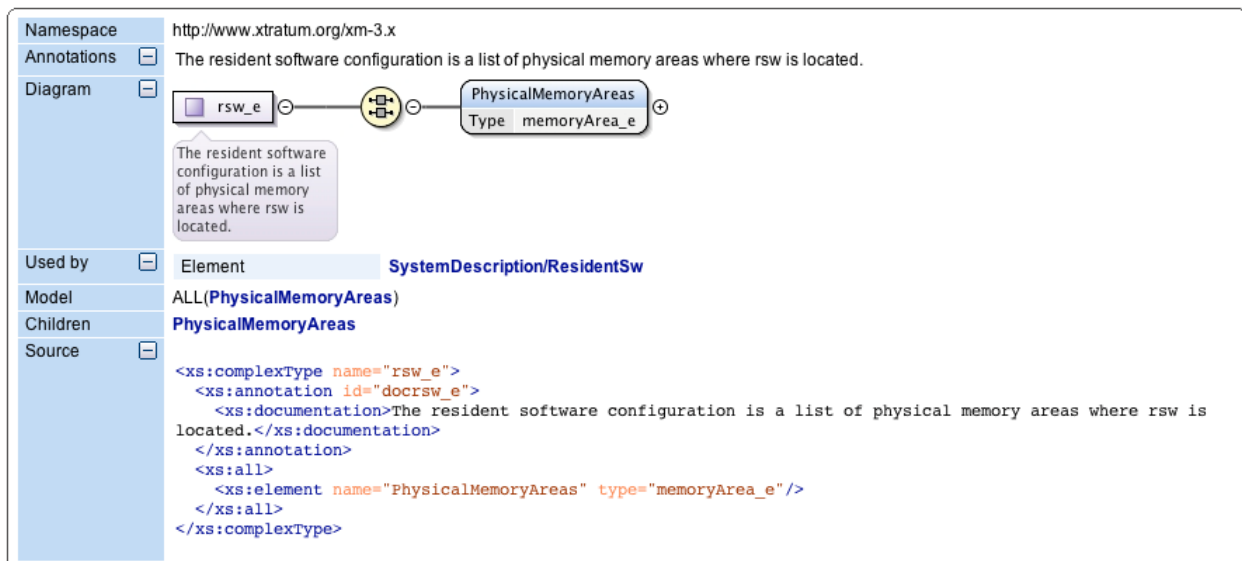


Figure 27: Resident Software component.

6.2.3.1 Element ResidentSw/PhysicalMemoryAreas

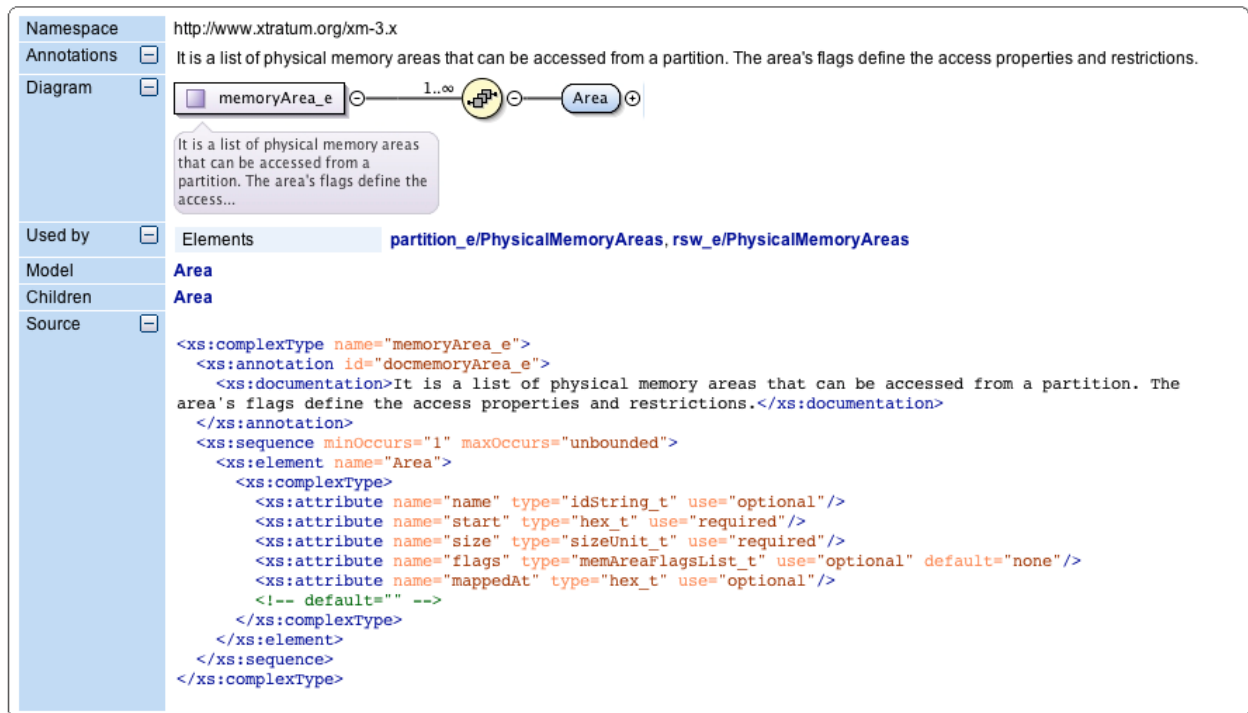


Figure 28: Resident Software memory.

6.2.4 Element PartitionTable

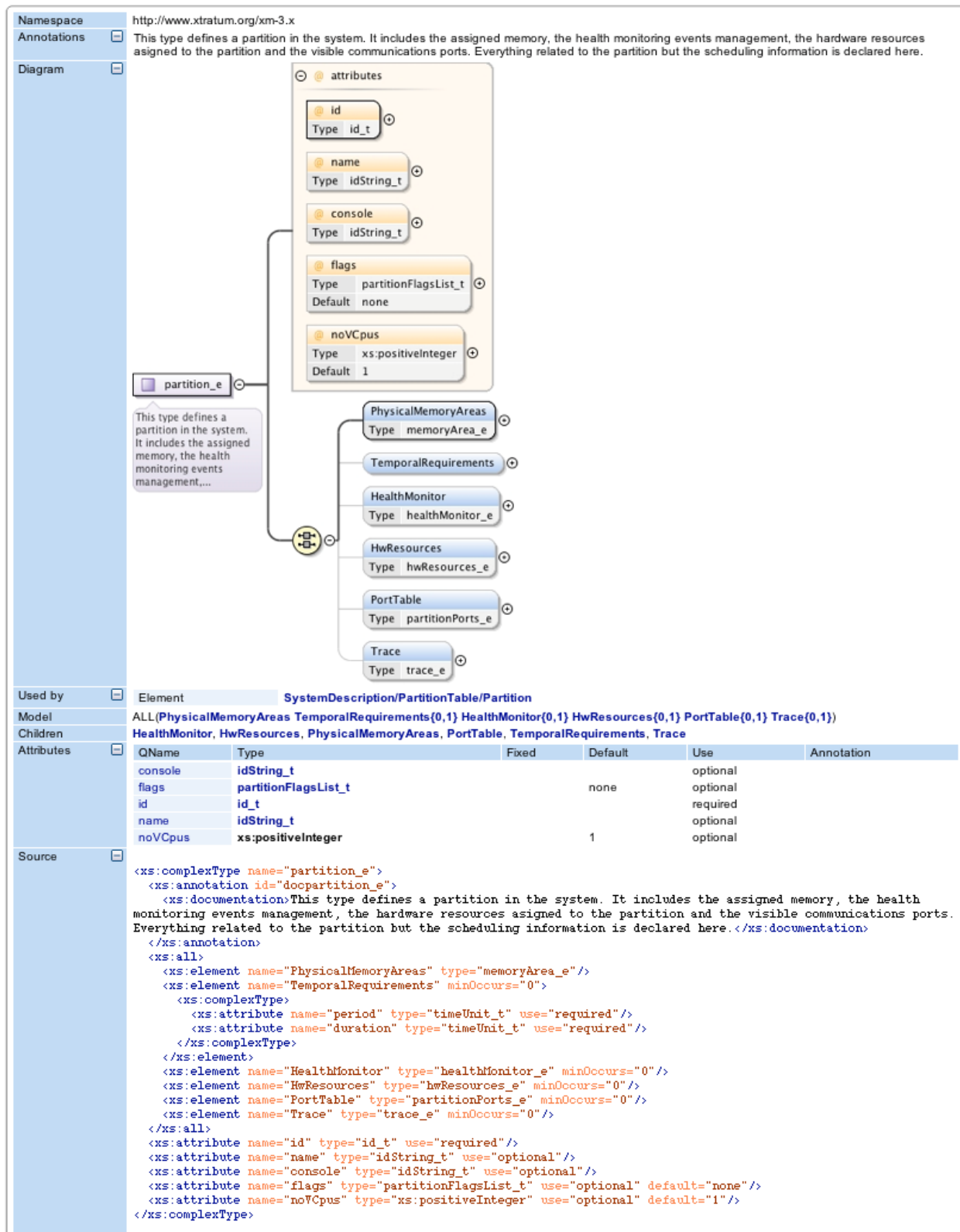


Figure 29: Partition component.

6.2.4.1 Element Partition/PhysicalMemoryAreas



Figure 30: Partition Memory.

6.2.4.2 Element Partition/HwResources

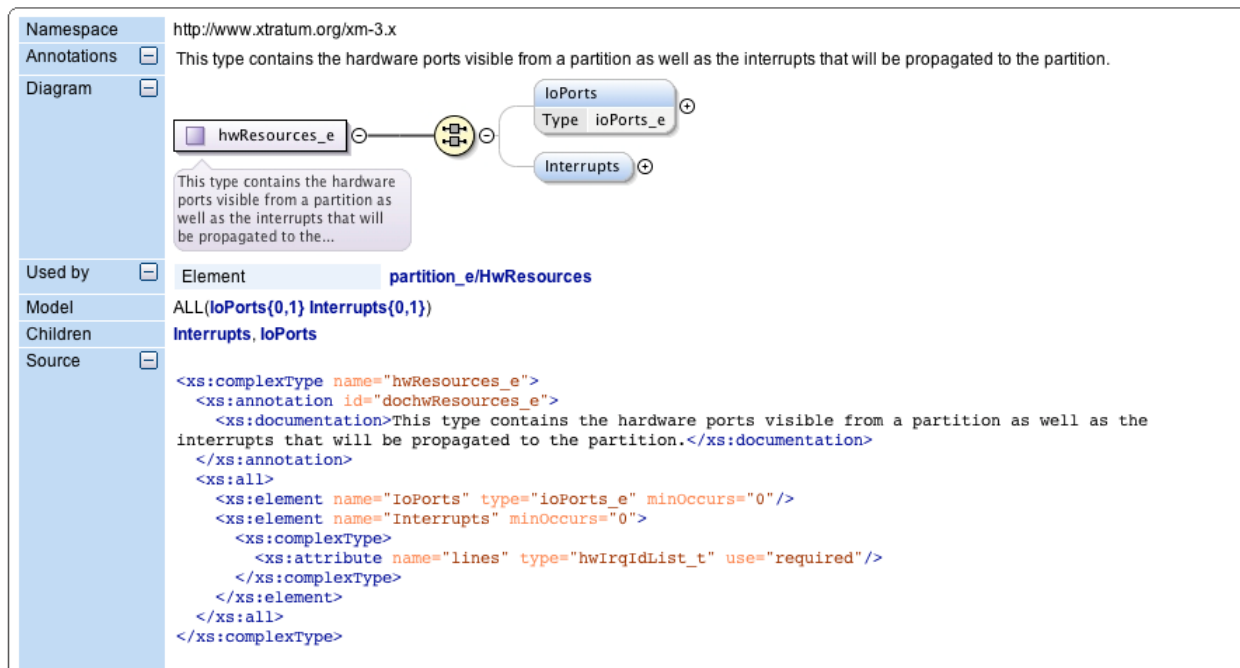


Figure 31: Partition hardware resources.

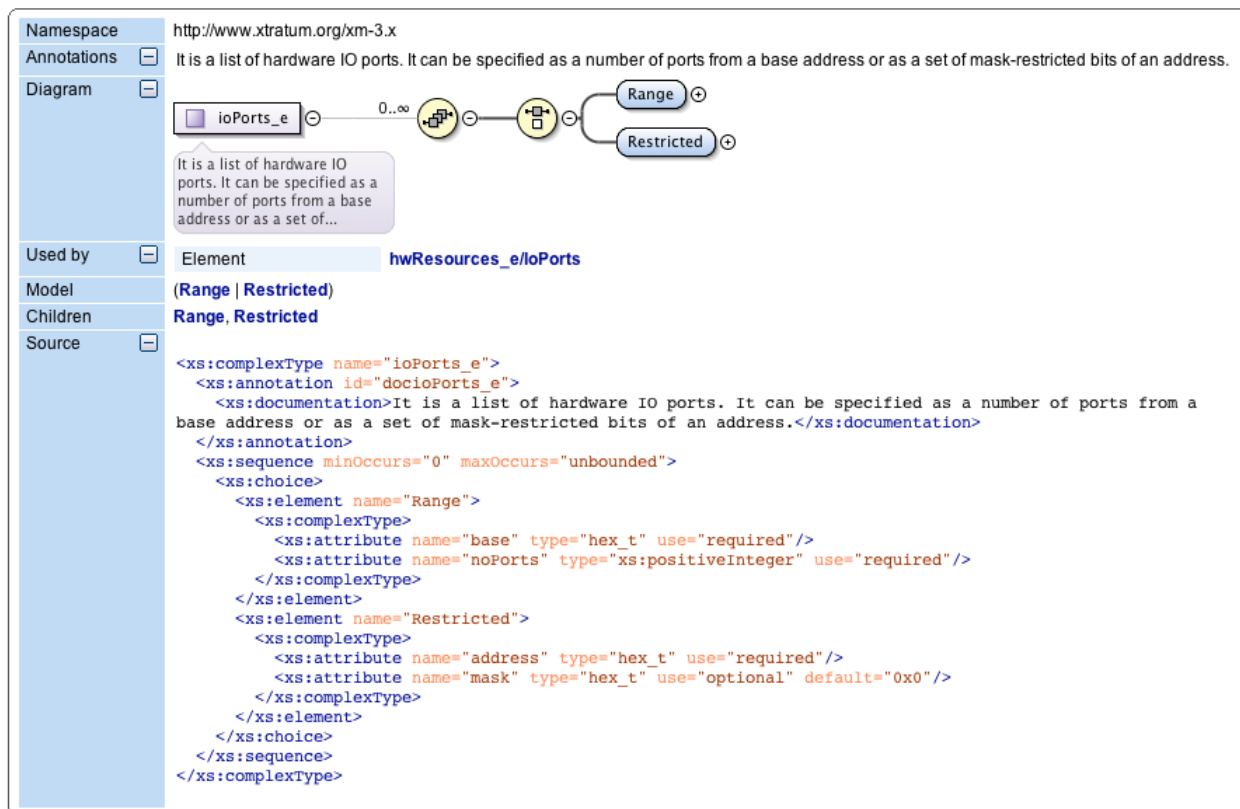


Figure 32: Partition I/O ports.

6.2.4.3 Element Partition/PortTable



Figure 33: Partition IPC ports.

6.2.5 Element Channels

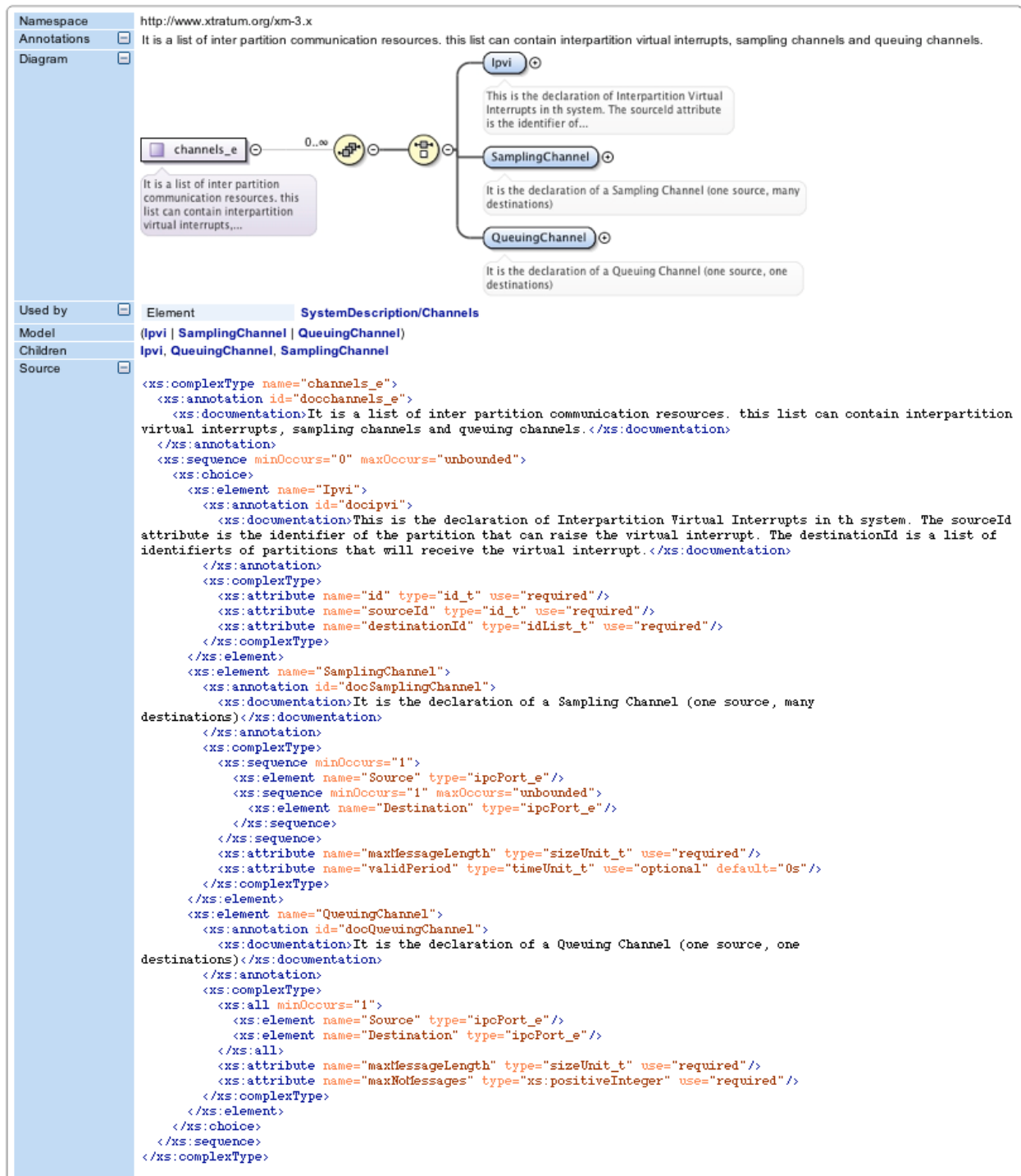


Figure 34: Channel component.

6.2.6 Basic types

Namespace	http://www.xratum.org/xm-3.x		
Annotations	It is an integer value used to identify objects in the hypervisor.		
Diagram	<p>It is an integer value used to identify objects in the hypervisor.</p> <p>Built-in derived type. The integer datatype is derived from decimal by fixing the value of fractionDigits to be 0. This...</p>		
Type	restriction of xs:integer		
Facets	MinInclusive	0	
Used by	Attributes	channels_e/lpvi/@id, channels_e/lpvi/@sourceId, fixedPrio_e/Partition/@id, fixedPrio_e/Partition/@vCpuId, ipcPort_e/@partitionId, partition_e/@id, plan_e/@id, plan_e/Slot/@id, plan_e/Slot/@partitionId, plan_e/Slot/@vCpuId, processor_e/@id	
Source	<pre><xs:simpleType name="id_t"> <xs:annotation id="docid_t"> <xs:documentation>It is an integer value used to identify objects in the hypervisor.</xs:documentation> </xs:annotation> <xs:restriction base="xs:integer"> <xs:minInclusive value="0"/> </xs:restriction> </xs:simpleType></pre>		

Namespace	http://www.xratum.org/xm-3.x		
Annotations	It is a list of hypervisor object identifiers.		
Diagram	<p>It is a list of hypervisor object identifiers.</p> <p>It is an integer value used to identify objects in the hypervisor.</p>		
Type	list of id_t		
Used by	Attribute	channels_e/lpvi/@destinationId	
Source	<pre><xs:simpleType name="idList_t"> <xs:annotation id="docidList_t"> <xs:documentation>It is a list of hypervisor object identifiers.</xs:documentation> </xs:annotation> <xs:list itemType="id_t"/> </xs:simpleType></pre>		

Namespace	http://www.xratum.org/xm-3.x		
Annotations	it is an hexadecimal value.		
Diagram	<p>it is an hexadecimal value.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>		
Type	restriction of xs:string		
Facets	Pattern	0x[0-9a-fA-F]+	
Used by	Attributes	devices_e/MemoryBlock/@start, ioPorts_e/Range/@base, ioPorts_e/Restricted/@address, ioPorts_e/Restricted/@mask, memoryArea_e/Area/@mappedAt, memoryArea_e/Area/@start, memoryLayout_e/Region/@start, trace_e/@bitmask	
Source	<pre><xs:simpleType name="hex_t"> <xs:annotation id="dochex_t"> <xs:documentation>it is an hexadecimal value.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:pattern value="0x[0-9a-fA-F]"/> </xs:restriction> </xs:simpleType></pre>		

Namespace	http://www.xratum.org/xm-3.x		
Annotations	It is a memory size value.		
Diagram	<p>It is a memory size value.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>		
Type	restriction of xs:string		
Facets	Pattern	[0-9]+([0-9]+)?([MK]?B)	
Used by	Attributes	channels_e/QueueingChannel/@maxLength, channels_e/SamplingChannel/@maxLength, devices_e/MemoryBlock/@size, hypMemoryArea_e/@size, memoryArea_e/Area/@size, memoryLayout_e/Region/@size	
Source	<pre><xs:simpleType name="sizeUnit_t"> <xs:annotation id="docsizeUnit_t"> <xs:documentation>It is a memory size value.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:pattern value="[0-9]+([0-9]+)?([MK]?B)"/> </xs:restriction> </xs:simpleType></pre>		

Namespace	http://www.xtratum.org/xm-3.x
Annotations	<input type="checkbox"/> It is a time duration value.
Diagram	
Type	restriction of xs:string
Facets	<input type="checkbox"/> Pattern [0-9]+(.[0-9]+)?([mu]?[sS])
Used by	<input type="checkbox"/> Attributes channels_e/SamplingChannel/@validPeriod, partition_e/TemporalRequirements/@duration, partition_e/TemporalRequirements/@period, plan_e/@majorFrame, plan_e/Slot/@duration, plan_e/Slot/@start
Source	<input type="checkbox"/> <pre> <xs:simpleType name="timeUnit_t"> <xs:annotation id="doctimeUnit_t"> <xs:documentation>It is a time duration value.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:pattern value="[0-9]+(.[0-9]+)?([mu]?[sS])" /> </xs:restriction> </xs:simpleType> </pre>

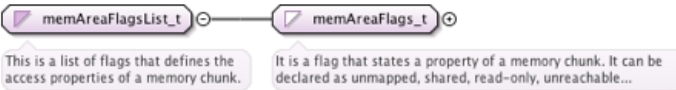
Namespace	http://www.xtratum.org/xm-3.x
Annotations	<input type="checkbox"/> It is a frequency value.
Diagram	
Type	restriction of xs:string
Facets	<input type="checkbox"/> Pattern [0-9]+(.[0-9]+)?([MKJ][Hh]z)
Used by	<input type="checkbox"/> Attribute processor_e/@frequency
Source	<input type="checkbox"/> <pre> <xs:simpleType name="freqUnit_t"> <xs:annotation id="docfreqUnit_t"> <xs:documentation>It is a frequency value.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:pattern value="[0-9]+(.[0-9]+)?([MKJ][Hh]z)" /> </xs:restriction> </xs:simpleType> </pre>

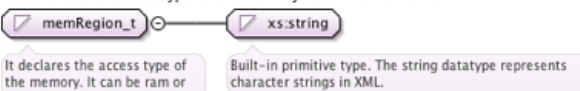
Namespace	http://www.xtratum.org/xm-3.x
Annotations	<input type="checkbox"/> It is a boolean value (yes, no, true, false).
Diagram	
Type	restriction of xs:string
Facets	<input type="checkbox"/> Enumeration yes Enumeration no Enumeration true Enumeration false
Used by	<input type="checkbox"/> Attribute healthMonitor_e/Event/@log
Source	<input type="checkbox"/> <pre> <xs:simpleType name="yntf_t"> <xs:annotation id="docyntf_t"> <xs:documentation>It is a boolean value (yes, no, true, false).</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="yes" /> <xs:enumeration value="no" /> <xs:enumeration value="true" /> <xs:enumeration value="false" /> </xs:restriction> </xs:simpleType> </pre>

Namespace	http://www.xtratum.org/xm-3.x				
Annotations	It defines the type of a communication's port: sampling, queuing.				
Diagram	<p>It defines the type of a communication's port: sampling, queuing.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>				
Type	restriction of xs:string				
Facets	<table border="1"> <tr> <td>Enumeration</td> <td>queuing</td> </tr> <tr> <td>Enumeration</td> <td>sampling</td> </tr> </table>	Enumeration	queuing	Enumeration	sampling
Enumeration	queuing				
Enumeration	sampling				
Used by	Attribute partitionPorts_e/Port/@type				
Source	<pre><xs:simpleType name="portType_t"> <xs:annotation id="docportType_t"> <xs:documentation>It defines the type of a communication's port: sampling, queuing.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="queuing"/> <xs:enumeration value="sampling"/> </xs:restriction> </xs:simpleType></pre>				

Namespace	http://www.xtratum.org/xm-3.x				
Annotations	It defines the direction of a communication port.				
Diagram	<p>It defines the direction of a communication port.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>				
Type	restriction of xs:string				
Facets	<table border="1"> <tr> <td>Enumeration</td> <td>source</td> </tr> <tr> <td>Enumeration</td> <td>destination</td> </tr> </table>	Enumeration	source	Enumeration	destination
Enumeration	source				
Enumeration	destination				
Used by	Attribute partitionPorts_e/Port/@direction				
Source	<pre><xs:simpleType name="direction_t"> <xs:annotation id="docdirection_t"> <xs:documentation>It defines the direction of a communication port.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="source"/> <xs:enumeration value="destination"/> </xs:restriction> </xs:simpleType></pre>				

Namespace	http://www.xtratum.org/xm-3.x																				
Annotations	It is a flag that states a property of a memory chunk. It can be declared as unmapped, shared, read-only, unreachable or rom. User specific flags are also supported.																				
Diagram	<p>It is a flag that states a property of a memory chunk. It can be declared as unmapped, shared, read-only, unreachable...</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>																				
Type	restriction of xs:string																				
Facets	<table border="1"> <tr><td>Enumeration</td><td>unmapped</td></tr> <tr><td>Enumeration</td><td>shared</td></tr> <tr><td>Enumeration</td><td>read-only</td></tr> <tr><td>Enumeration</td><td>uncacheable</td></tr> <tr><td>Enumeration</td><td>rom</td></tr> <tr><td>Enumeration</td><td>flag0</td></tr> <tr><td>Enumeration</td><td>flag1</td></tr> <tr><td>Enumeration</td><td>flag2</td></tr> <tr><td>Enumeration</td><td>flag3</td></tr> <tr><td>Enumeration</td><td>none</td></tr> </table>	Enumeration	unmapped	Enumeration	shared	Enumeration	read-only	Enumeration	uncacheable	Enumeration	rom	Enumeration	flag0	Enumeration	flag1	Enumeration	flag2	Enumeration	flag3	Enumeration	none
Enumeration	unmapped																				
Enumeration	shared																				
Enumeration	read-only																				
Enumeration	uncacheable																				
Enumeration	rom																				
Enumeration	flag0																				
Enumeration	flag1																				
Enumeration	flag2																				
Enumeration	flag3																				
Enumeration	none																				
Source	<pre><xs:simpleType name="memAreaFlags_t"> <xs:annotation id="docmemAreaFlags_t"> <xs:documentation>It is a flag that states a property of a memory chunk. It can be declared as unmapped, shared, read-only, unreachable or rom. User specific flags are also supported.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="unmapped"/> <xs:enumeration value="shared"/> <xs:enumeration value="read-only"/> <xs:enumeration value="uncacheable"/> <xs:enumeration value="rom"/> <xs:enumeration value="flag0"/> <xs:enumeration value="flag1"/> <xs:enumeration value="flag2"/> <xs:enumeration value="flag3"/> <xs:enumeration value="none"/> </xs:restriction> </xs:simpleType></pre>																				

Namespace	http://www.xtraturn.org/xm-3.x
Annotations	<input type="checkbox"/> This is a list of flags that defines the access properties of a memory chunk.
Diagram	<input type="checkbox"/>  <p>This is a list of flags that defines the access properties of a memory chunk.</p> <p>It is a flag that states a property of a memory chunk. It can be declared as unmapped, shared, read-only, unreachable...</p>
Type	list of memAreaFlags_t
Used by	<input type="checkbox"/> Attributes hypMemoryArea_e/@flags, memoryArea_e/Area/@flags
Source	<input type="checkbox"/> <pre> <xs:simpleType name="memAreaFlagsList_t"> <xs:annotation id="docmemAreaFlagsList_t"> <xs:documentation>This is a list of flags that defines the access properties of a memory chunk.</xs:documentation> </xs:annotation> <xs:list itemType="memAreaFlags_t" /> </xs:simpleType> </pre>

Namespace	http://www.xtraturn.org/xm-3.x				
Annotations	<input type="checkbox"/> It declares the access type of the memory. It can be ram or rom.				
Diagram	<input type="checkbox"/>  <p>It declares the access type of the memory. It can be ram or rom.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>				
Type	restriction of xs:string				
Facets	<input type="checkbox"/> <table> <tr> <td>Enumeration</td><td>ram</td></tr> <tr> <td>Enumeration</td><td>rom</td></tr> </table>	Enumeration	ram	Enumeration	rom
Enumeration	ram				
Enumeration	rom				
Used by	<input type="checkbox"/> Attribute memoryLayout_e/Region/@type				
Source	<input type="checkbox"/> <pre> <xs:simpleType name="memRegion_t"> <xs:annotation id="docmemRegion_t"> <xs:documentation>It declares the access type of the memory. It can be ram or rom.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="ram"/> <xs:enumeration value="rom"/> </xs:restriction> </xs:simpleType> </pre>				

Namespace	http://www.xtratum.org/xm-3.x																																																					
Annotations	<input checked="" type="checkbox"/> This type enumerates the health monitoring events supported by the x86 processor.																																																					
Diagram	<input checked="" type="checkbox"/>																																																					
Type	restriction of xs:string																																																					
Facets	<input checked="" type="checkbox"/> <table border="1"> <tr><td>Enumeration</td><td>XM_HM_EV_INTERNAL_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_SYSTEM_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_PARTITION_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_WATCHDOG_TIMER</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_FP_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_MEM_PROTECTION</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_UNEXPECTED_TRAP</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_DIVIDE_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_DEBUG</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_NMI_INTERRUPT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_BREAKPOINT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_OVERFLOW</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_BOUND_RANGE_EXCEEDED</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_INVALID_OPCODE</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_DEVICE_NOT_AVAILABLE</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_DOUBLE_FAULT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_COPROCESSOR_SEGMENT_OVERRUN</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_INVALID_TSS</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_SEGMENT_NOT_PRESENT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_STACK_FAULT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_GENERAL_PROTECTION</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_PAGE_FAULT</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_X87_FPU_ERROR</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_ALIGNMENT_CHECK</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_MACHINE_CHECK</td></tr> <tr><td>Enumeration</td><td>XM_HM_EV_X86_SIMD_FLOATING_POINT</td></tr> </table>		Enumeration	XM_HM_EV_INTERNAL_ERROR	Enumeration	XM_HM_EV_SYSTEM_ERROR	Enumeration	XM_HM_EV_PARTITION_ERROR	Enumeration	XM_HM_EV_WATCHDOG_TIMER	Enumeration	XM_HM_EV_FP_ERROR	Enumeration	XM_HM_EV_MEM_PROTECTION	Enumeration	XM_HM_EV_UNEXPECTED_TRAP	Enumeration	XM_HM_EV_X86_DIVIDE_ERROR	Enumeration	XM_HM_EV_X86_DEBUG	Enumeration	XM_HM_EV_X86_NMI_INTERRUPT	Enumeration	XM_HM_EV_X86_BREAKPOINT	Enumeration	XM_HM_EV_X86_OVERFLOW	Enumeration	XM_HM_EV_X86_BOUND_RANGE_EXCEEDED	Enumeration	XM_HM_EV_X86_INVALID_OPCODE	Enumeration	XM_HM_EV_X86_DEVICE_NOT_AVAILABLE	Enumeration	XM_HM_EV_X86_DOUBLE_FAULT	Enumeration	XM_HM_EV_X86_COPROCESSOR_SEGMENT_OVERRUN	Enumeration	XM_HM_EV_X86_INVALID_TSS	Enumeration	XM_HM_EV_X86_SEGMENT_NOT_PRESENT	Enumeration	XM_HM_EV_X86_STACK_FAULT	Enumeration	XM_HM_EV_X86_GENERAL_PROTECTION	Enumeration	XM_HM_EV_X86_PAGE_FAULT	Enumeration	XM_HM_EV_X86_X87_FPU_ERROR	Enumeration	XM_HM_EV_X86_ALIGNMENT_CHECK	Enumeration	XM_HM_EV_X86_MACHINE_CHECK	Enumeration	XM_HM_EV_X86_SIMD_FLOATING_POINT
Enumeration	XM_HM_EV_INTERNAL_ERROR																																																					
Enumeration	XM_HM_EV_SYSTEM_ERROR																																																					
Enumeration	XM_HM_EV_PARTITION_ERROR																																																					
Enumeration	XM_HM_EV_WATCHDOG_TIMER																																																					
Enumeration	XM_HM_EV_FP_ERROR																																																					
Enumeration	XM_HM_EV_MEM_PROTECTION																																																					
Enumeration	XM_HM_EV_UNEXPECTED_TRAP																																																					
Enumeration	XM_HM_EV_X86_DIVIDE_ERROR																																																					
Enumeration	XM_HM_EV_X86_DEBUG																																																					
Enumeration	XM_HM_EV_X86_NMI_INTERRUPT																																																					
Enumeration	XM_HM_EV_X86_BREAKPOINT																																																					
Enumeration	XM_HM_EV_X86_OVERFLOW																																																					
Enumeration	XM_HM_EV_X86_BOUND_RANGE_EXCEEDED																																																					
Enumeration	XM_HM_EV_X86_INVALID_OPCODE																																																					
Enumeration	XM_HM_EV_X86_DEVICE_NOT_AVAILABLE																																																					
Enumeration	XM_HM_EV_X86_DOUBLE_FAULT																																																					
Enumeration	XM_HM_EV_X86_COPROCESSOR_SEGMENT_OVERRUN																																																					
Enumeration	XM_HM_EV_X86_INVALID_TSS																																																					
Enumeration	XM_HM_EV_X86_SEGMENT_NOT_PRESENT																																																					
Enumeration	XM_HM_EV_X86_STACK_FAULT																																																					
Enumeration	XM_HM_EV_X86_GENERAL_PROTECTION																																																					
Enumeration	XM_HM_EV_X86_PAGE_FAULT																																																					
Enumeration	XM_HM_EV_X86_X87_FPU_ERROR																																																					
Enumeration	XM_HM_EV_X86_ALIGNMENT_CHECK																																																					
Enumeration	XM_HM_EV_X86_MACHINE_CHECK																																																					
Enumeration	XM_HM_EV_X86_SIMD_FLOATING_POINT																																																					
Used by	<input checked="" type="checkbox"/> Attribute healthMonitor_e/Event/@name																																																					
Source	<input checked="" type="checkbox"/> <pre> <xs:simpleType name="hmString_t"> <xs:annotation id="dochmString_t"> <xs:documentation>This type enumerates the health monitoring events supported by the x86 processor.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="XM_HM_EV_INTERNAL_ERROR"/> <xs:enumeration value="XM_HM_EV_SYSTEM_ERROR"/> <xs:enumeration value="XM_HM_EV_PARTITION_ERROR"/> <xs:enumeration value="XM_HM_EV_WATCHDOG_TIMER"/> <xs:enumeration value="XM_HM_EV_FP_ERROR"/> <xs:enumeration value="XM_HM_EV_MEM_PROTECTION"/> <xs:enumeration value="XM_HM_EV_UNEXPECTED_TRAP"/> <xs:enumeration value="XM_HM_EV_X86_DIVIDE_ERROR"/> <xs:enumeration value="XM_HM_EV_X86_DEBUG"/> <xs:enumeration value="XM_HM_EV_X86_NMI_INTERRUPT"/> <xs:enumeration value="XM_HM_EV_X86_BREAKPOINT"/> <xs:enumeration value="XM_HM_EV_X86_OVERFLOW"/> <xs:enumeration value="XM_HM_EV_X86_BOUND_RANGE_EXCEEDED"/> <xs:enumeration value="XM_HM_EV_X86_INVALID_OPCODE"/> <xs:enumeration value="XM_HM_EV_X86_DEVICE_NOT_AVAILABLE"/> <xs:enumeration value="XM_HM_EV_X86_DOUBLE_FAULT"/> <xs:enumeration value="XM_HM_EV_X86_COPROCESSOR_SEGMENT_OVERRUN"/> <xs:enumeration value="XM_HM_EV_X86_INVALID_TSS"/> <xs:enumeration value="XM_HM_EV_X86_SEGMENT_NOT_PRESENT"/> <xs:enumeration value="XM_HM_EV_X86_STACK_FAULT"/> <xs:enumeration value="XM_HM_EV_X86_GENERAL_PROTECTION"/> <xs:enumeration value="XM_HM_EV_X86_PAGE_FAULT"/> <xs:enumeration value="XM_HM_EV_X86_X87_FPU_ERROR"/> <xs:enumeration value="XM_HM_EV_X86_ALIGNMENT_CHECK"/> <xs:enumeration value="XM_HM_EV_X86_MACHINE_CHECK"/> <xs:enumeration value="XM_HM_EV_X86_SIMD_FLOATING_POINT"/> </xs:restriction> </xs:simpleType> </pre>																																																					

Namespace	http://www.xtratum.org/xm-3.x																						
Annotations	<input checked="" type="checkbox"/> This type enumerates the health monitoring actions supported by the hypervisor.																						
Diagram	<input checked="" type="checkbox"/> <p>This type enumerates the health monitoring actions supported by the hypervisor.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>																						
Type	restriction of xs:string																						
Facets	<input checked="" type="checkbox"/> <table> <tr><td>Enumeration</td><td>XM_HM_AC_IGNORE</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_SHUTDOWN</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_PARTITION_COLD_RESET</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_PARTITION_WARM_RESET</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_HYPERVISOR_COLD_RESET</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_HYPERVISOR_WARM_RESET</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_SUSPEND</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_PARTITION_HALT</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_HYPERVISOR_HALT</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_PROPAGATE</td></tr> <tr><td>Enumeration</td><td>XM_HM_AC_SWITCH_TO_MAINTENANCE</td></tr> </table>	Enumeration	XM_HM_AC_IGNORE	Enumeration	XM_HM_AC_SHUTDOWN	Enumeration	XM_HM_AC_PARTITION_COLD_RESET	Enumeration	XM_HM_AC_PARTITION_WARM_RESET	Enumeration	XM_HM_AC_HYPERVISOR_COLD_RESET	Enumeration	XM_HM_AC_HYPERVISOR_WARM_RESET	Enumeration	XM_HM_AC_SUSPEND	Enumeration	XM_HM_AC_PARTITION_HALT	Enumeration	XM_HM_AC_HYPERVISOR_HALT	Enumeration	XM_HM_AC_PROPAGATE	Enumeration	XM_HM_AC_SWITCH_TO_MAINTENANCE
Enumeration	XM_HM_AC_IGNORE																						
Enumeration	XM_HM_AC_SHUTDOWN																						
Enumeration	XM_HM_AC_PARTITION_COLD_RESET																						
Enumeration	XM_HM_AC_PARTITION_WARM_RESET																						
Enumeration	XM_HM_AC_HYPERVISOR_COLD_RESET																						
Enumeration	XM_HM_AC_HYPERVISOR_WARM_RESET																						
Enumeration	XM_HM_AC_SUSPEND																						
Enumeration	XM_HM_AC_PARTITION_HALT																						
Enumeration	XM_HM_AC_HYPERVISOR_HALT																						
Enumeration	XM_HM_AC_PROPAGATE																						
Enumeration	XM_HM_AC_SWITCH_TO_MAINTENANCE																						
Used by	<input checked="" type="checkbox"/> Attribute healthMonitor_e/Event/@action																						
Source	<input checked="" type="checkbox"/> <pre> <xs:simpleType name="hmAction_t"> <xs:annotation id="dochmAction_t"> <xs:documentation>This type enumerates the health monitoring actions supported by the hypervisor.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="XM_HM_AC_IGNORE" /> <xs:enumeration value="XM_HM_AC_SHUTDOWN" /> <xs:enumeration value="XM_HM_AC_PARTITION_COLD_RESET" /> <xs:enumeration value="XM_HM_AC_PARTITION_WARM_RESET" /> <xs:enumeration value="XM_HM_AC_HYPERVISOR_COLD_RESET" /> <xs:enumeration value="XM_HM_AC_HYPERVISOR_WARM_RESET" /> <xs:enumeration value="XM_HM_AC_SUSPEND" /> <xs:enumeration value="XM_HM_AC_PARTITION_HALT" /> <xs:enumeration value="XM_HM_AC_HYPERVISOR_HALT" /> <xs:enumeration value="XM_HM_AC_PROPAGATE" /> <xs:enumeration value="XM_HM_AC_SWITCH_TO_MAINTENANCE" /> </xs:restriction> </xs:simpleType> </pre>																						

Namespace	http://www.xtratum.org/xm-3.x								
Annotations	<input checked="" type="checkbox"/> It specifies one supported event type.								
Diagram	<input checked="" type="checkbox"/> <p>It specifies one supported event type.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>								
Type	restriction of xs:string								
Facets	<input checked="" type="checkbox"/> <table> <tr><td>Enumeration</td><td>HYP_IRQS</td></tr> <tr><td>Enumeration</td><td>HYP_HCALLS</td></tr> <tr><td>Enumeration</td><td>HYP_SCHED</td></tr> <tr><td>Enumeration</td><td>HYP_HM</td></tr> </table>	Enumeration	HYP_IRQS	Enumeration	HYP_HCALLS	Enumeration	HYP_SCHED	Enumeration	HYP_HM
Enumeration	HYP_IRQS								
Enumeration	HYP_HCALLS								
Enumeration	HYP_SCHED								
Enumeration	HYP_HM								
Source	<input checked="" type="checkbox"/> <pre> <xs:simpleType name="traceHyp_t"> <xs:annotation id="doctraceHyp_t"> <xs:documentation>It specifies one supported event type.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="HYP_IRQS" /> <xs:enumeration value="HYP_HCALLS" /> <xs:enumeration value="HYP_SCHED" /> <xs:enumeration value="HYP_HM" /> </xs:restriction> </xs:simpleType> </pre>								

Namespace	http://www.xtratum.org/xm-3.x
Annotations	<input checked="" type="checkbox"/> This is a list of the event types that will be traced.
Diagram	<input checked="" type="checkbox"/> <p>This is a list of the event types that will be traced.</p> <p>It specifies one supported event type.</p>
Type	list of traceHyp_t
Used by	<input checked="" type="checkbox"/> Attribute traceHyp_e/@bitmask
Source	<input checked="" type="checkbox"/> <pre> <xs:simpleType name="traceHypList_t"> <xs:annotation id="doctraceHypList_t"> <xs:documentation>This is a list of the event types that will be traced.</xs:documentation> </xs:annotation> <xs:list itemType="traceHyp_t" /> </xs:simpleType> </pre>

Namespace	http://www.xratum.org/xm-3.x	
Annotations	This is a flag enabling a partition's role in the system with associated access rights: system, fp, none.	
Diagram	<p>This is a flag enabling a partition's role in the system with associated access rights: system, fp, none.</p> <p>Built-in primitive type. The string datatype represents character strings in XML.</p>	
Type	restriction of xs:string	
Facets	Enumeration	system
	Enumeration	fp
	Enumeration	none
Source	<pre> <xs:simpleType name="partitionFlags_t"> <xs:annotation id="docpartitionFlags_t"> <xs:documentation>This is a flag enabling a partition's role in the system with associated access rights: system, fp, none.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="system" /> <xs:enumeration value="fp" /> <xs:enumeration value="none" /> </xs:restriction> </xs:simpleType> </pre>	

Namespace	http://www.xratum.org/xm-3.x	
Annotations	List of partition flags which define the role of the partition in the system.	
Diagram	<p>List of partition flags which define the role of the partition in the system.</p> <p>This is a flag enabling a partition's role in the system with associated access rights: system, fp, none.</p>	
Type	list of partitionFlags_t	
Used by	Attribute	partition_e/@flags
Source	<pre> <xs:simpleType name="partitionFlagsList_t"> <xs:annotation id="docpartitionFlagsList_t"> <xs:documentation>List of partition flags which define the role of the partition in the system.</xs:documentation> </xs:annotation> <xs:list itemType="partitionFlags_t" /> </xs:simpleType> </pre>	

Namespace	http://www.xratum.org/xm-3.x	
Annotations	It is an integer value used to identify interrupts.	
Diagram	<p>It is an integer value used to identify interrupts.</p> <p>Built-in derived type. The integer datatype is derived from decimal by fixing the value of fractionDigits to be 0. This...</p>	
Type	restriction of xs:integer	
Facets	MaxExclusive	16
	MinInclusive	0
Source	<pre> <xs:simpleType name="hwIrqId_t"> <xs:annotation id="dochwIrqId_t"> <xs:documentation>It is an integer value used to identify interrupts.</xs:documentation> </xs:annotation> <xs:restriction base="xs:integer"> <xs:minInclusive value="0" /> <xs:maxExclusive value="16" /> </xs:restriction> </xs:simpleType> </pre>	

Namespace	http://www.xratum.org/xm-3.x	
Annotations	It is a list of interrupts.	
Diagram	<p>It is a list of interrupts.</p> <p>It is an integer value used to identify interrupts.</p>	
Type	list of hwIrqId_t	
Used by	Attribute	hwResources_e/Interrupts/@lines
Source	<pre> <xs:simpleType name="hwIrqIdList_t"> <xs:annotation id="dochwIrqIdList_t"> <xs:documentation>It is a list of interrupts.</xs:documentation> </xs:annotation> <xs:list itemType="hwIrqId_t" /> </xs:simpleType> </pre>	

Namespace	http://www.xtratum.org/xm-3.x				
Annotations	<input type="checkbox"/> This type enumerates the supported special processor features.				
Diagram	<input type="checkbox"/>				
Type	restriction of xs:string				
Facets	<input type="checkbox"/> <table> <tr> <td>Enumeration</td><td>XM_CPU_LEON2_WA1</td></tr> <tr> <td>Enumeration</td><td>none</td></tr> </table>	Enumeration	XM_CPU_LEON2_WA1	Enumeration	none
Enumeration	XM_CPU_LEON2_WA1				
Enumeration	none				
Source	<input type="checkbox"/> <pre> <xs:simpleType name="processorFeatures_t"> <xs:annotation id="docprocessorFeatures_t"> <xs:documentation>This type enumerates the supported special processor features.</xs:documentation> </xs:annotation> <xs:restriction base="xs:string"> <xs:enumeration value="XM_CPU_LEON2_WA1" /> <xs:enumeration value="none" /> </xs:restriction> </xs:simpleType> </pre>				

Namespace	http://www.xtratum.org/xm-3.x
Annotations	<input type="checkbox"/> It is a list of special features of a physical processor.
Diagram	<input type="checkbox"/>
Type	list of processorFeatures_t
Used by	<input type="checkbox"/> Attribute processor_e/@features
Source	<input type="checkbox"/> <pre> <xs:simpleType name="processorFeaturesList_t"> <xs:annotation id="docprocessorFeaturesList_t"> <xs:documentation>It is a list of special features of a physical processor.</xs:documentation> </xs:annotation> <xs:list itemType="processorFeatures_t" /> </xs:simpleType> </pre>

7 Secure State and Secure Operations

The Secure Kernel Partition Profile [6] defines the rationale for *secure state* as:

Definition: “*secure state*” is based on two separate properties:

- (A) that the TSF is capable of enforcing the security policy (i.e., its own data and mechanisms are intact); and
- (B) that exported resources are correctly separated (e.g., application data, and related descendants and copies, are associated with the correct data).

7.1 Secure State

Property (A) states that the "secure state" is related to the integrity and coherence of the internal data and mechanisms. Internal data of Virtualization Layer (VLayer) can be considered:

- configuration vector as binary representation of the configuration file used to define the system that has been validated, compiled and included in the final system container. In execution, this configuration vector resides in the hypervisor (kernel) address space (not accessible by subjects) in a memory area that is write protected. Additionally, a digest (cryptographic hash function) is applied to the configuration vector which is added to it. At any moment, the VLayer can perform the digest of the configuration vector and validate its integrity.
- internal variables: state of the VLayer. VLayer status is formed by the tuple

$$\langle \text{PART}^{\text{current}}, \text{PLAN}^{\text{current}}, \text{SLOT}^{\text{current}}, \text{CLOCK}^{\text{current}} \rangle$$
 that refers to the current partition under execution, current plan, current slot and current time. The coherence of these variables is fundamental in the virtualization layer operation. The pair $\langle \text{PLAN}^{\text{current}}, \text{CLOCK}^{\text{current}} \rangle$ determines the slot and the partition in the configuration vector that should correspond with the current partition and current slot. Once the state is validated, through the configuration vector is possible to determine the exported resources (Memory areas, FPU, IRQs, etc.) and operations that should be applied.
- the processor registers: MMU registers, interrupt vector (IV), mode processor status (MPS), IO protection and FPU control.
- channels: the consistency of the channel data structures can be determined with respect to the maximum values (message length and number of messages) defined in the configuration vector.
- stacks: The VLayer maintains one stack for the own VLayer operations and one stack per partition which is used when the VLayer executes a hypercall for a specific partition. The limits of these stacks can be validated. It is important to note that each partition maintains its own stack in the user space when the partition is executing operations at user level.

As for the configuration vector, a digest of the VLayer core is applied at deployment phase which is included in the deployment and can be validated. This validation will permit to affirm that the VLayer code has not been corrupted.

An important aspect with respect to the secure state is the limited preemptability of XtratuM. It implies that during any VLayer operation, it can be interrupted only at specific points that are very

well identified. This limitation has a strong impact in the simplicity of the VLayer design (less complex) and the security (more secure).

Property (B) is related to the isolation properties (spatial and temporal) of subjects. The VLayer has been designed so that the individual effects of operations that violate the policy are privileged operations (operations over virtualized resources) or by means of hypercalls with not allowed parameters (i.e. reset the system by a subject not authorized). In the first case, the forbidden operation generates a trap that is captured by the VLayer and generates a Health Monitor event which involves an HM action with the goal of maintaining the VLayer in the "secure state" (i.e. the subject can be halted (disabled) or restarted according the action defined in the configuration file (XM_CF).

In the second case, the hypercall with non-allowed parameters, the VLayer performs an exhaustive validation of the parameters according the configuration vector and refuse the operation (returns a code error in the hypercall to the subject invoking the hypercall).

7.2 Insecure state

When the conditions stated previously cannot be validated, the VLayer is in an "insecure state". The following situations can determine that the VLayer is in an "insecure state":

- configuration vector pollution. The digest of the configuration vector does not match with the correct value.
- VLayer code pollution. The digest of the VLayer code does not match with the correct value.
- Deviation of the internal state. The tupla $\langle \text{PART}^{\text{current}}, \text{PLAN}^{\text{current}}, \text{SLOT}^{\text{current}}, \text{CLOCK}^{\text{current}} \rangle$ is not coherent with $\langle \text{PART}^{\text{XM_CF}}, \text{PLAN}^{\text{current}}, \text{SLOT}^{\text{XM_CF}}, \text{CLOCK}^{\text{current}} \rangle$ obtained from the configuration vector.
- Access to non exported resources for a partition. A partition can perform an operation to an exported resource that has not been defined in the configuration vector. Note that if a partition requests an operation on a non exported resource the hypercall should return a code error.
- Limits exceeds. Stacks and channels data structures exceeds the limit values established in the configuration vector.
- Underlying hardware: clock, timers, memory protection mechanisms, IO protection mechanisms, FPU protection mechanisms.

Any of these situations determine that the VLayer is not in a "secure state". In these cases, it is not possible to change to a "secure state" and the system has to be reset.

7.3 Trustability enforcement

Some aspects should be revisited with respect to the VLayer:

- It is non pre-emptable. When any of the entry-points is invoked, it is executed with disabled interrupts returning the control to a partition.
- It has three entry points and one return point to partition.

- All exported resources are defined in the configuration vector.
- Only the hypervisor can access to the privileged registers and virtualized services.
- Internal code of partitions is not relevant from the hypervisor point of view

Additionally, it is assumed that the underlying hardware is trusted. It means that the internal processor registers will work properly if they are used in the correct way.

Based on the trustiness of the hardware mechanisms, XtratuM permits to extend the trustability including the hypervisor level.

7.4 Test for secure states

As result of the previous analysis the state of the VLayer could be evaluated at different levels. SKPP (Separation Kernel Protection Profile) defines different types of tests to achieve a secure state at boot time and during the execution of the partitions.

7.4.1 Abstract machine test (AMT)

In general the AMT refers to the proper operation of the hardware platform on which a VLayer is running. These tests permit to consider that the underlaying hardware is trusted and extends it to the VLayer. It is executed at boot time in order to guarantee a secure boot. It includes:

- Timers test
- Protection mechanisms test: MMU, privileged operation, IO protection, FPU control.
- Memory Read and Write: This test can read/write/read portions of memory to ensure the values written remain unchanged.
- Memory Separation and Protection: to ensure that user space programs cannot read and write to areas of memory that is protected.
- Privileged Instructions: it ensures that the enforcement of the property that privileged instructions should only be in supervisor mode is still in effect.

7.4.2 Basic platform tests

Basic tests rely on basic properties of the hypervisor and the trust enforcement from the trusted hardware. They should be executed each time a partition is scheduled in order to guarantee a secure partition operation. It includes:

- Validation of the internal variables related to the VLayer state
- Processor registers: MMU registers, interrupt vector (IV), mode processor status (MPS), IO protection and FPU control.
- Stack limits
- Monotonic clock

These tests require low computation resources. They are executed each time the VLayer performs a partition context switch.

7.4.3 Maintenance tests

Self tests are related with the self evaluation of the XtratuM security functions (XSF) with respect to some expected correct operation. It includes:

- Stack limits: VLayer and partition's stacks.
- Configuration vector (perform a digest of the current configuration vector and compare it with the deployed digest).
- VLayer code (perform a digest of the XtratuM code and compare it with the deployed digest).
- Channel limits evaluation
- Partition code pollution evaluation. This is an operation that should be performed by each partition. XtratuM only saw at the deployment phase a binary file of the partition without distinction of code and data. The partition knows the internal segments and could perform some evaluation if the code has been polluted, limits of internal data structures, etc.

These tests should be executed during a maintenance phase of the system.

8 DREAMS Abstraction Layer (DRAL)

This section describes the software architecture supported by DREAMS, which involves several applications with different levels of criticality. It details the execution environment and the services provided to support the application execution.

The software architecture is built on top of a DREAMS node that manages the entire tile including one or more processor cores.

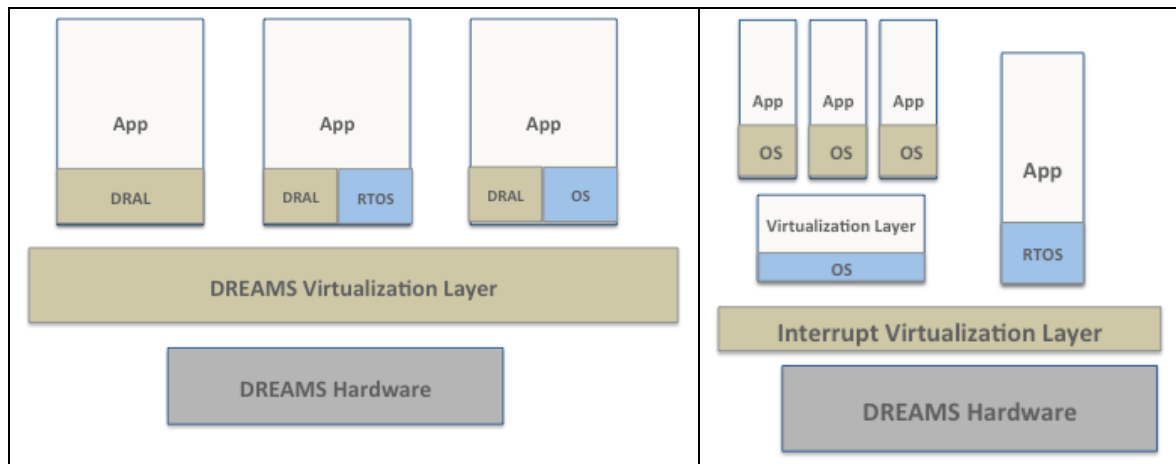


Figure 1: Software architectures

In order to support mixed-criticality applications, the DREAMS software architecture is composed by:

- Virtualization layer: It is a software layer that provides hardware virtualization to the applications. Two different approaches are considered in DREAMS depending on the application constraints.
 - Partitioning kernel: It provides virtualization of the hardware resources by defining a set of services that are used by the partitions to access the virtualized resources. The partitioning kernel provides spatial and temporal isolation to the partitions.
 - Interrupt Virtualization layer: This layer virtualizes the Host OS interrupts and is only introduced when KVM hypervisor is used. The main objective is to take hardware interrupts control away from Host OS and handle them in a thin layer, so as to preserve timing guarantees for the RTOS. Thus, an interrupt virtualization layer (ADEOS or similar) is introduced below the Host OS and real-time partition to prioritize the RTOS.
- Partitions: A partition is the execution unit in the DREAMS architecture. It provides the basic infrastructure to execute an application. Different partitions are supported in the DREAMS architecture.
 - Basic single-thread application to be executed near a native hardware
 - Multi-thread real-time applications to be executed on top of a real-time operating system
 - Multi process applications to be executed on top of a full featured operating system
 - Multi-partition applications to be executed on top of a operating system that provides the ability to build virtualized multiple process applications.

8.1 DRAL

A complete DRAL specification is presented in a different document named *D2.3.1 – Annex. DREAMS Abstraction Layer (DRAL) Specification*.

8.1.1 System Management Services

System Management Services refer to the services that a partition can invoke to get the status of the virtualization layer or perform actions on it.

Services are:

Name	Description	Constraints
DRAL_GET_SYSTEM_STATUS	Returns the status of the virtualization layer. The result is a data structure that provides some information related to the current status. In the case of interrupt virtualization, this service will set the configuration details of such a layer, for instance, interrupt masking, peripheral binding/unbinding, etc.	System
DRAL_SET_SYSTEM_MODE	Provides to a partition the ability to change the status of the virtualization layer. Actions to be invoked are: - Perform a cold reset on the system. As result of this invocation, the system is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked. - Perform a warm reset on the system. As result of this invocation, the system is reset and boots. The reset counter is increased. - Perform a system halt. As result of this invocation, the system is halted. A physical reset is required to restart the system.	System

8.1.2 Partition Management Services

Partition Management Services refer to the services that a partition can invoke to get its own status or other partition status or perform actions on them.

Services are:

Name	Description	Constraints
DRAL_GET_PARTITION_ID	Access to the partition identifier.	Normal
DRAL_GET_PARTITION_ID_BY_NAME	Access to the partition identifier from the partition name.	System /Normal
DRAL_GET_PARTITION_STATUS	Returns the status of a partition. The result is a data structure that provides some information related to the current partition status.	System /Normal
DRAL_SET_PARTITION_MODE	It provides to a partition the ability to change its own status or the status of other partition. Actions to be invoked are: - Perform a cold reset on a partition. As result of this invocation, the partition is reset and boots. A counter	System /Normal

	<p>informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked.</p> <ul style="list-style-type: none"> - Perform a warm reset on a partition. As result of this invocation, the partition is reset and boots. The reset counter is increased. - Perform a partition halt. As result of this invocation, the partition is halted. - Perform a partition suspend. As result of this invocation, the partition is suspended. - Perform a partition resume. As result of this invocation, the partition is resumed. <p>In the case of interrupt virtualization, this service will set the configuration details of such a layer, for instance, interrupt masking, peripheral binding/unbinding, etc.</p>	
--	--	--

8.1.3 Process Management

These services are provided by the GuestOS.

8.1.4 Time Management Services

Time Management Services refer to the services that a partition can invoke to get time information or set timers.

Time can be global or local. Global time is referred to a monotonic clock of the system. Local time is referred to a partition clock that runs when the partition is executed. Timers can be set taking as reference the global or the local time.

Services are:

Name	Description	Constraints
DRAL_GET_TIME	Get the current time (global or local).	Normal
DRAL_SET_TIMER	Set a timer referred to the global or local clock.	Normal

8.1.5 Inter-Partition Communication Services

A partition can send/receive messages to/from other partitions using sampling or queuing ports.

Services are:

Name	Description	Constraints
DRAL_CREATE_SAMPLING_PORT	Creates a sampling port.	Normal
DRAL_WRITE_SAMPLING_MESSAGE	Writes a message in a sampling port.	Normal
DRAL_READ_SAMPLING_MESSAGE	Reads a message in a sampling port.	Normal
DRAL_CREATE_QUEUEING_PORT	Creates a sampling port.	Normal
DRAL_SEND_QUEUEING_MESSAGE	Sends a message in a queuing port.	Normal
DRAL_RECEIVE_QUEUEING_MESSAGE	Receives a message in a queuing port.	Normal
DRAL_GET_QUEUEING_PORT_STATUS	Gets the status of a queuing port.	Normal
DRAL_CLEAR_QUEUEING_PORT	Removes all messages in a queuing port.	Normal

8.1.6 Intra-Partition Communication

These services are provided by the GuestOS.

8.1.7 Scheduling Services

A partition is scheduled under the virtualization layer policy. It is relevant for the partition to get the information related to its own schedule. On the other hand, a partition can be interested in define local schedules for other partitions in spare slots. How to deal with spare slots and dynamic allocation of resources will be discussed in WP4.

GPOS sub-partitions created by KVM will also use these services to get scheduling policy details. In this use case the RTOS system partition will be able to force a scheduling policy on partitions that offer virtualization features (Linux/KVM partition).

Services are:

Name	Description	Constraints
DRAL_GET_PARTITION_SCHEDULE	Gets the information of the partition schedule in a MAF.	Normal
DRAL_GET_PARTITION_SCHEDULE_STATUS	Gets the information related to the current execution slot.	Normal
DRAL_SET_MODULE_SCHEDULE	Requests for a schedule plan change.	System
DRAL_GET_MODULE_SCHEDULE_STATUS	Gets the current schedule plan status.	Normal

8.1.8 Monitoring Services (Health Monitor)

A partition can raise health monitor (HM) events to the virtualization layer. These HM events are detected and generated by the application or the partition runtime. The events that the partition can raise are:

- APPLICATION ERROR: An error in the application.
- DEADLINE MISSED: A deadline miss has been detected.
- NUMERIC ERROR: The application has detected a numeric error.
- STACK OVERFLOW: The partition detects a stack overflow.
- MEMORY VIOLATION: The partition detects an illegal memory access.

Services are:

Name	Description	Constraints
DRAL_GET_ERROR_STATUS	Permits to the partition to access to the reported errors.	Normal
DRAL_RAISE_APPLICATION_ERROR	The partition raises an HM event that will be handled by the virtualization layer	Normal

8.1.9 Configuration services

The following table summarizes what constitutes configurations services, i.e. all services that allow for reconfiguration of the system:

Name	Description	Constraints
DRAL_SET_MODULE_SCHEDULE	Requests for a schedule plan change.	System

DRAL_SET_PARTITION_MODE	<p>It provides to a partition the ability to change its own status or the status of other partition. Actions to be invoked are:</p> <ul style="list-style-type: none">- Perform a cold reset on a partition. As result of this invocation, the partition is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked.- Perform a warm reset on a partition. As result of this invocation, the partition is reset and boots. The reset counter is increased.- Perform a partition halt. As result of this invocation, the partition is halted.- Perform a partition suspend. As result of this invocation, the partition is suspended.- Perform a partition resume. As result of this invocation, the partition is resumed.	System /Normal
DRAL_SET_SYSTEM_MODE	<p>Provides to a partition the ability to change the status of the virtualization layer. Actions to be invoked are:</p> <ul style="list-style-type: none">- Perform a cold reset on the system. As result of this invocation, the system is reset and boots. A counter informs about the number of consecutive warm resets have been produced. This counter is zeroed when the cold reset is invoked.- Perform a warm reset on the system. As result of this invocation, the system is reset and boots. The reset counter is increased.- Perform a system halt. As result of this invocation, the system is halted. A physical reset is required to restart the system.	System

9 Bibliography

- [1] Barham, P. a. (2003). Xen and the art of virtualization. In P. o. principles (Ed.). (pp. 164-177). New York: ACM
- [2] A. Addisu, L. George, V. Sciandra, and M. Agueh. Mixed criticality scheduling applied to jpeg2000 video streaming over wireless multimedia sensor networks. In Proc. WMC, RTSS , pages 55–60, 2013.
- [3] Common criteria for information technology security evaluation, July 2009. Version 3.1 Rev. 3. CCMB- 2009-07-001.
- [4] Common criteria for information technology security evaluation, July 2009. Version 3.1 Rev. 3. CCMB- 2009-07-002.
- [5] Common criteria for information technology security evaluation, July 2009. Version 3.1 Rev. 3. CCMB- 2009-07-003.
- [6] Information Assurance Directorate. SKPP: Separation kernel protection profile, Version 1.03 29 June 2007.
- [7] Xilinx. Zynq-7000 All Programmable SoC – Technical Reference Manual. UG585 v1.8.1. September 19, 2014.
- [8] Fent Innovative Software Solutions, “XtratuM Hypervisor for ARM CORTEX-A9 SMP – User Manual.”, fnts-xm-um-arm-121b, March, 2014.
- [9] Gosain, Yashu and Palanichamy, Prushothaman. Xilinx. White Paper: TrustZone Technology Support in Zynq-7000 All Programmable SoCs. May 20, 2014.
- [10] Fent Innovative Software Solutions, “XtratuM Hypervisor for ARM CORTEX-A9 SMP – Reference Manual.”, fnts-xm-rm-arm-122a, March, 2014.

APPENDIX 1

- X86-XSD (XML Schema Definition):

```
<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.xtratum.org/xm-3.x"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://www.xtratum.org/xm-3.x"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- Basic types definition -->
  <xs:simpleType name="id_t">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="idString_t">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="hwIrqId_t">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxExclusive value="16"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="hwIrqIdList_t">
    <xs:list itemType="hwIrqId_t"/>
  </xs:simpleType>
  <xs:simpleType name="idList_t">
    <xs:list itemType="id_t"/>
  </xs:simpleType>
  <xs:simpleType name="hex_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="0x[0-9a-fA-F]+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="version_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+.[0-9]+.[0-9]+"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="freqUnit_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+.(.[0-9]+)?([MK][Hh]z)"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="processorFeatures_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="XM_CPU_LEON2_WA1"/>
      <xs:enumeration value="none"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="processorFeaturesList_t">
    <xs:list itemType="processorFeatures_t"/>
  </xs:simpleType>
  <xs:simpleType name="partitionFlags_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="system"/>
      <xs:enumeration value="fp"/>
      <xs:enumeration value="none"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="partitionFlagsList_t">
    <xs:list itemType="partitionFlags_t"/>
  </xs:simpleType>
  <xs:simpleType name="sizeUnit_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+.(.[0-9]+)?([MK]?B)"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="timeUnit_t">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]+.(.[0-9]+)?([mu]?[sS])"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="traceHyp_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="HYP_IRQS"/>
      <xs:enumeration value="HYP_HCALLS"/>
      <xs:enumeration value="HYP_SCHED"/>
      <xs:enumeration value="HYP_HM"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="traceHypList_t">
    <xs:list itemType="traceHyp_t"/>
  </xs:simpleType>
  <!--@ \void{<track id="xml-list-hm-events">} @-->
  <xs:simpleType name="hmString_t">
    <xs:restriction base="xs:string">
      <xs:enumeration value="XM_HM_EV_INTERNAL_ERROR"/>
      <xs:enumeration value="XM_HM_EV_SYSTEM_ERROR"/>
      <xs:enumeration value="XM_HM_EV_PARTITION_ERROR"/>
      <xs:enumeration value="XM_HM_EV_WATCHDOG_TIMER"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

```

<xs:enumeration value="XM_HM_EV_FP_ERROR"/>
<xs:enumeration value="XM_HM_EV_MEM_PROTECTION"/>
<xs:enumeration value="XM_HM_EV_UNEXPECTED_TRAP"/>
<xs:enumeration value="XM_HM_EV_X86_DIVIDE_ERROR"/>
<xs:enumeration value="XM_HM_EV_X86_DEBUG"/>
<xs:enumeration value="XM_HM_EV_X86_NMI_INTERRUPT"/>
<xs:enumeration value="XM_HM_EV_X86_BREAKPOINT"/>
<xs:enumeration value="XM_HM_EV_X86_OVERFLOW"/>
<xs:enumeration value="XM_HM_EV_X86_BOUND_RANGE_EXCEEDED"/>
<xs:enumeration value="XM_HM_EV_X86_INVALID_OPCODE"/>
<xs:enumeration value="XM_HM_EV_X86_DEVICE_NOT_AVAILABLE"/>
<xs:enumeration value="XM_HM_EV_X86_DOUBLE_FAULT"/>
<xs:enumeration value="XM_HM_EV_X86_COPROCESSOR_SEGMENT_OVERRUN"/>
<xs:enumeration value="XM_HM_EV_X86_INVALID_TSS"/>
<xs:enumeration value="XM_HM_EV_X86_SEGMENT_NOT_PRESENT"/>
<xs:enumeration value="XM_HM_EV_X86_STACK_FAULT"/>
<xs:enumeration value="XM_HM_EV_X86_GENERAL_PROTECTION"/>
<xs:enumeration value="XM_HM_EV_X86_PAGE_FAULT"/>
<xs:enumeration value="XM_HM_EV_X86_X87_FPU_ERROR"/>
<xs:enumeration value="XM_HM_EV_X86_ALIGNMENT_CHECK"/>
<xs:enumeration value="XM_HM_EV_X86_MACHINE_CHECK"/>
<xs:enumeration value="XM_HM_EV_X86_SIMD_FLOATING_POINT"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="hmAction_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="XM_HM_AC_IGNORE"/>
    <xs:enumeration value="XM_HM_AC_SHUTDOWN"/>
    <xs:enumeration value="XM_HM_AC_PARTITION_COLD_RESET"/>
    <xs:enumeration value="XM_HM_AC_PARTITION_WARM_RESET"/>
    <xs:enumeration value="XM_HM_AC_HYPERVISOR_COLD_RESET"/>
    <xs:enumeration value="XM_HM_AC_HYPERVISOR_WARM_RESET"/>
    <xs:enumeration value="XM_HM_AC_SUSPEND"/>
    <xs:enumeration value="XM_HM_AC_PARTITION_HALT"/>
    <xs:enumeration value="XM_HM_AC_HYPERVISOR_HALT"/>
    <xs:enumeration value="XM_HM_AC_PROPAGATE"/>
    <xs:enumeration value="XM_HM_AC_SWITCH_TO_MAINTENANCE"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="memAreaFlags_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="unmapped"/>
    <xs:enumeration value="shared"/>
    <xs:enumeration value="read-only"/>
    <xs:enumeration value="uncacheable"/>
    <xs:enumeration value="rom"/>
    <xs:enumeration value="flag0"/>
    <xs:enumeration value="flag1"/>
    <xs:enumeration value="flag2"/>
    <xs:enumeration value="flag3"/>
    <xs:enumeration value="iommu" />
    <xs:enumeration value="none"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="memAreaFlagsList_t">
  <xs:list itemType="memAreaFlags_t"/>
</xs:simpleType>
<xs:simpleType name="memRegion_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ram"/>
    <xs:enumeration value="rom"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="portType_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="queuing"/>
    <xs:enumeration value="sampling"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="direction_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="source"/>
    <xs:enumeration value="destination"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="yntf_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="yes"/>
    <xs:enumeration value="no"/>
    <xs:enumeration value="true"/>
    <xs:enumeration value="false"/>
  </xs:restriction>
</xs:simpleType>
<!-- End Types -->
<!-- Elements -->
<!-- Hypervisor -->
<xs:complexType name="hypervisor_e">
  <xs:all>
    <xs:element name="PhysicalMemoryArea" type="hypMemoryArea_e"/>
    <xs:element name="HealthMonitor" type="healthMonitor_e" minOccurs="0"/>
    <xs:element name="Trace" type="traceHyp_e" minOccurs="0"/>
  </xs:all>
  <xs:attribute name="console" type="idString_t" use="optional"/>
  <xs:attribute name="healthMonitorDevice" type="idString_t" use="optional"/>
</xs:complexType>
<!-- Rsw -->
<xs:complexType name="rsw_e">
  <xs:all>
    <xs:element name="PhysicalMemoryAreas" type="memoryArea_e"/>
  </xs:all>
</xs:complexType>
<!-- Partition -->

```

```
<xs:complexType name="partition_e">
  <xs:all>
    <xs:element name="PhysicalMemoryAreas" type="memoryArea_e"/>
    <xs:element name="TemporalRequirements" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="period" type="timeUnit_t" use="required"/>
        <xs:attribute name="duration" type="timeUnit_t" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="HealthMonitor" type="healthMonitor_e" minOccurs="0"/>
    <xs:element name="HwResources" type="hwResources_e" minOccurs="0"/>
    <xs:element name="PortTable" type="partitionPorts_e" minOccurs="0"/>
    <xs:element name="Trace" type="trace_e" minOccurs="0"/>
  </xs:all>
  <xs:attribute name="id" type="id_t" use="required"/>
  <xs:attribute name="name" type="idString_t" use="optional"/>
  <xs:attribute name="console" type="idString_t" use="optional"/>
  <xs:attribute name="flags" type="partitionFlagsList_t" use="optional" default="none"/>
  <xs:attribute name="noVCpus" type="xs:positiveInteger" use="optional" default="1"/>
</xs:complexType>
<!-- Trace -->
<xs:complexType name="trace_e">
  <xs:attribute name="device" type="idString_t" use="required"/>
  <xs:attribute name="bitmask" type="hex_t" use="required"/>
</xs:complexType>
<xs:complexType name="traceHyp_e">
  <xs:attribute name="device" type="idString_t" use="required"/>
  <xs:attribute name="bitmask" type="traceHypList_t" use="required"/>
</xs:complexType>
<!-- Communication Ports -->
<xs:complexType name="partitionPorts_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Port">
      <xs:complexType>
        <xs:attribute name="name" type="idString_t" use="required"/>
        <xs:attribute name="direction" type="direction_t" use="required"/>
        <xs:attribute name="type" type="portType_t" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Channels -->
<xs:complexType name="channels_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="Ipvi">
        <xs:complexType>
          <xs:attribute name="id" type="id_t" use="required"/>
          <xs:attribute name="sourceId" type="id_t" use="required"/>
          <xs:attribute name="destinationId" type="idList_t" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="SamplingChannel">
        <xs:complexType>
          <xs:sequence minOccurs="1">
            <xs:element name="Source" type="ipcPort_e"/>
            <xs:sequence minOccurs="1" maxOccurs="unbounded">
              <xs:element name="Destination" type="ipcPort_e"/>
            </xs:sequence>
          </xs:sequence>
          <xs:attribute name="maxMessageLength" type="sizeUnit_t" use="required"/>
          <xs:attribute name="validPeriod" type="timeUnit_t" use="optional" default="0s"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="QueuingChannel">
        <xs:complexType>
          <xs:all minOccurs="1">
            <xs:element name="Source" type="ipcPort_e"/>
            <xs:element name="Destination" type="ipcPort_e"/>
          </xs:all>
          <xs:attribute name="maxMessageLength" type="sizeUnit_t" use="required"/>
          <xs:attribute name="maxNoMessages" type="xs:positiveInteger" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<!-- Devices -->
<xs:complexType name="devices_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="MemoryBlock" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="required"/>
          <xs:attribute name="start" type="hex_t" use="required"/>
          <xs:attribute name="size" type="sizeUnit_t" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Uart" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="required"/>
          <xs:attribute name="id" type="idString_t" use="required"/>
          <xs:attribute name="baudRate" type="xs:positiveInteger" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Vga" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Null" minOccurs="0">
        <xs:complexType>
```

```
<xs:attribute name="name" type="idString_t" use="optional"/>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:sequence>
</xs:complexType>
<!-- IPC Port -->
<xs:complexType name="ipcPort_e">
  <xs:attribute name="partitionId" type="id_t" use="required"/>
  <xs:attribute name="partitionName" type="idString_t" use="optional"/>
  <xs:attribute name="portName" type="idString_t" use="required"/>
</xs:complexType>
<!-- Hw Description -->
<xs:complexType name="hwDescription_e">
  <xs:sequence>
    <xs:element name="MemoryLayout" type="memoryLayout_e"/>
    <xs:element name="ProcessorTable">
      <xs:complexType>
        <xs:sequence minOccurs="1" maxOccurs="256">
          <xs:element name="Processor" type="processor_e"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Devices" type="devices_e"/>
  </xs:sequence>
</xs:complexType>
<!-- Processor -->
<xs:complexType name="processor_e">
  <!-- <xs:all>
  <xs:element name="CyclicPlanTable" type="cyclicPlan_e"/>
  </xs:all> -->
  <xs:choice>
    <xs:element name="CyclicPlanTable" type="cyclicPlan_e"/>
    <xs:element name="FixedPriority" type="fixedPrio_e"/>
  </xs:choice>
  <xs:attribute name="id" type="id_t" use="required"/>
  <xs:attribute name="frequency" type="freqUnit_t" use="optional"/>
  <xs:attribute name="features" type="processorFeaturesList_t" use="optional" default="none"/>
</xs:complexType>
<!-- HwResource -->
<xs:complexType name="hwResources_e">
  <xs:all>
    <xs:element name="IoPorts" type="ioPorts_e" minOccurs="0"/>
    <xs:element name="Interrupts" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="lines" type="hwIrqIdList_t" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:all>
</xs:complexType>
<!-- Io Ports -->
<xs:complexType name="ioPorts_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="Range">
        <xs:complexType>
          <xs:attribute name="base" type="hex_t" use="required"/>
          <xs:attribute name="noPorts" type="xs:positiveInteger" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Restricted">
        <xs:complexType>
          <xs:attribute name="address" type="hex_t" use="required"/>
          <xs:attribute name="mask" type="hex_t" use="optional" default="0x0"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<!-- Fixed priority -->
<xs:complexType name="fixedPrio_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Partition">
      <xs:complexType>
        <xs:attribute name="id" type="id_t" use="required"/>
        <xs:attribute name="vCpuId" type="id_t" use="optional" default="0"/>
        <xs:attribute name="priority" type="xs:positiveInteger" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- CyclicPlan -->
<xs:complexType name="cyclicPlan_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Plan" type="plan_e"/>
  </xs:sequence>
</xs:complexType>
<!-- Plan -->
<xs:complexType name="plan_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Slot">
      <xs:complexType>
        <xs:attribute name="id" type="id_t" use="required"/>
        <xs:attribute name="start" type="timeUnit_t" use="required"/>
        <xs:attribute name="duration" type="timeUnit_t" use="required"/>
        <xs:attribute name="partitionId" type="id_t" use="required"/>
        <xs:attribute name="vCpuId" type="id_t" use="optional" default="0"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="idString_t" use="optional"/>
  <xs:attribute name="id" type="id_t" use="required"/>

```

```

    <xs:attribute name="majorFrame" type="timeUnit_t" use="optional" default="0s"/>
  </xs:complexType>
  <!-- Health Monitor -->
  <xs:complexType name="healthMonitor_e">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="Event">
        <xs:complexType>
          <xs:attribute name="name" type="hmString_t" use="required"/>
          <xs:attribute name="action" type="hmAction_t" use="required"/>
          <xs:attribute name="log" type="yntf_t" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Memory Layout -->
  <xs:complexType name="memoryLayout_e">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="Region">
        <xs:complexType>
          <xs:attribute name="type" type="memRegion_t" use="required"/>
          <xs:attribute name="start" type="hex_t" use="required"/>
          <xs:attribute name="size" type="sizeUnit_t" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Hypervisor Memory Area -->
  <xs:complexType name="hypMemoryArea_e">
    <xs:attribute name="size" type="sizeUnit_t" use="required"/>
    <xs:attribute name="flags" type="memAreaFlagsList_t" use="optional"/>
  </xs:complexType>
  <!-- Memory Area -->
  <xs:complexType name="memoryArea_e">
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="Area">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="optional"/>
          <xs:attribute name="start" type="hex_t" use="required"/>
          <xs:attribute name="size" type="sizeUnit_t" use="required"/>
          <xs:attribute name="flags" type="memAreaFlagsList_t" use="optional" default="none"/>
          <xs:attribute name="mappedAt" type="hex_t" use="optional"/>
          <!-- default="" -->
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <!-- Root Element -->
  <xs:element name="SystemDescription">
    <xs:complexType>
      <xs:all>
        <xs:element name="HwDescription" type="hwDescription_e"/>
        <xs:element name="XMHypervisor" type="hypervisor_e"/>
        <xs:element name="ResidentSw" type="rsw_e" minOccurs="0"/>
        <xs:element name="PartitionTable">
          <xs:complexType>
            <xs:sequence maxOccurs="unbounded">
              <xs:element name="Partition" type="partition_e"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Channels" type="channels_e" minOccurs="0"/>
      </xs:all>
      <xs:attribute name="version" type="version_t" use="required"/>
      <xs:attribute name="name" type="idString_t" use="required"/>
    </xs:complexType>
  </xs:element>
  <!-- End Root Element -->
  <!-- Elements -->
</xs:schema>

```

- ARM-XSD (XML Schema Definition):

```

<?xml version="1.0"?>
<xs:schema targetNamespace="http://www.xtratum.org/xm-3.x"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.xtratum.org/xm-3.x"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <!-- Basic types definition -->
  <xs:simpleType name="id_t">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="idString_t">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="hwIrqId_t">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxExclusive value="96"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="hwIrqIdList_t">
    <xs:list itemType="hwIrqId_t"/>
  </xs:simpleType>

```

```
</xs:simpleType>
<xs:simpleType name="idList_t">
  <xs:list itemType="id_t"/>
</xs:simpleType>
<xs:simpleType name="hex_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="0x[0-9a-fA-F] +"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="version_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]+.[0-9]+.[0-9] +"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="freqUnit_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]+(.[0-9]+)?([MK][Hh]z)"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="processorFeatures_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="XM_CPU_LEON2_WA1"/>
    <xs:enumeration value="none"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="discipline_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="FIFO"/>
    <xs:enumeration value="PRIORITY"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="processorFeaturesList_t">
  <xs:list itemType="processorFeatures_t"/>
</xs:simpleType>
<xs:simpleType name="partitionFlags_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="system"/>
    <xs:enumeration value="fp"/>
    <xs:enumeration value="boot"/>
    <xs:enumeration value="icache_disabled"/>
    <xs:enumeration value="dcache_disabled"/>
    <xs:enumeration value="none"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="partitionFlagsList_t">
  <xs:list itemType="partitionFlags_t"/>
</xs:simpleType>
<xs:simpleType name="sizeUnit_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]+(.[0-9]+)?([MK]?B)"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="timeUnit_t">
  <xs:restriction base="xs:string">
    <xs:pattern value="[0-9]+(.[0-9]+)?([mu]?[sS])"/>
  </xs:restriction>
</xs:simpleType>
<!--@ \void{<track id="xml-list-hm-events">} @-->
<xs:simpleType name="hmString_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="XM_HM_EV_INTERNAL_ERROR"/>
    <xs:enumeration value="XM_HM_EV_UNEXPECTED_TRAP"/>
    <xs:enumeration value="XM_HM_EV_PARTITION_ERROR"/>
    <xs:enumeration value="XM_HM_EV_PARTITION_INTEGRITY"/>
    <xs:enumeration value="XM_HM_EV_MEM_PROTECTION"/>
    <xs:enumeration value="XM_HM_EV_OVERRUN"/>
    <xs:enumeration value="XM_HM_EV_SCHED_ERROR"/>
    <xs:enumeration value="XM_HM_EV_WATCHDOG_TIMER"/>
    <xs:enumeration value="XM_HM_EV_INCOMPATIBLE_INTERFACE"/>
    <xs:enumeration value="XM_HM_EV_ARM_UNDEF_INSTR"/>
    <xs:enumeration value="XM_HM_EV_ARM_PREFETCH_ABORT"/>
    <xs:enumeration value="XM_HM_EV_ARM_DATA_ABORT"/>
    <xs:enumeration value="XM_HM_EV_ARM_DATA_ALIGNMENT_FAULT"/>
    <xs:enumeration value="XM_HM_EV_ARM_DATA_BACKGROUND_FAULT"/>
    <xs:enumeration value="XM_HM_EV_ARM_DATA_PERMISSION_FAULT"/>
    <xs:enumeration value="XM_HM_EV_ARM_INSTR_ALIGNMENT_FAULT"/>
    <xs:enumeration value="XM_HM_EV_ARM_INSTR_BACKGROUND_FAULT"/>
    <xs:enumeration value="XM_HM_EV_ARM_INSTR_PERMISSION_FAULT"/>
  </xs:restriction>
</xs:simpleType>
<!--@ \void{<track id="xml-list-hm-events">} @-->
<!--@ \void{<track id="xml-list-hm-actions">} @-->
<xs:simpleType name="hmAction_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="XM_HM_AC_IGNORE"/>
    <xs:enumeration value="XM_HM_AC_SHUTDOWN"/>
    <xs:enumeration value="XM_HM_AC_PARTITION_COLD_RESET"/>
    <xs:enumeration value="XM_HM_AC_PARTITION_WARM_RESET"/>
    <xs:enumeration value="XM_HM_AC_HYPERVISOR_COLD_RESET"/>
    <xs:enumeration value="XM_HM_AC_HYPERVISOR_WARM_RESET"/>
    <xs:enumeration value="XM_HM_AC_SUSPEND"/>
    <xs:enumeration value="XM_HM_AC_HALT"/>
    <xs:enumeration value="XM_HM_AC_PROPAGATE"/>
    <xs:enumeration value="XM_HM_AC_SWITCH_TO_MAINTENANCE"/>
  </xs:restriction>
</xs:simpleType>
<!--@ \void{<track id="xml-list-hm-actions">} @-->
<xs:simpleType name="memAreaFlags_t">
  <xs:restriction base="xs:string">
    <xs:enumeration value="unmapped"/>
    <xs:enumeration value="read-only"/>
    <xs:enumeration value="uncacheable"/>
  </xs:restriction>
</xs:simpleType>
```



```

        <xs:enumeration value="rom"/>
        <xs:enumeration value="flag0"/>
        <xs:enumeration value="flag1"/>
        <xs:enumeration value="flag2"/>
        <xs:enumeration value="flag3"/>
        <xs:enumeration value="none"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="memAreaFlagsList_t">
    <xs:list itemType="memAreaFlags_t"/>
</xs:simpleType>
<xs:simpleType name="slotFlags_t">
    <xs:restriction base="xs:string">
        <xs:enumeration value="periodStart"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="slotFlagsList_t">
    <xs:list itemType="slotFlags_t"/>
</xs:simpleType>
<xs:simpleType name="memRegion_t">
    <xs:restriction base="xs:string">
        <xs:enumeration value="sdram"/>
        <xs:enumeration value="stram"/>
        <xs:enumeration value="rom"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="portType_t">
    <xs:restriction base="xs:string">
        <xs:enumeration value="queuing"/>
        <xs:enumeration value="sampling"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="direction_t">
    <xs:restriction base="xs:string">
        <xs:enumeration value="source"/>
        <xs:enumeration value="destination"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="yntf_t">
    <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
        <xs:enumeration value="true"/>
        <xs:enumeration value="false"/>
    </xs:restriction>
</xs:simpleType>
<!-- End Types -->
<!-- Elements -->
<!-- Hypervisor -->
<xs:complexType name="hypervisor_e">
    <xs:all>
        <xs:element name="PhysicalMemoryArea" type="hypMemoryArea_e"/>
        <xs:element name="HealthMonitor" type="healthMonitor_e" minOccurs="0" />
    </xs:all>
    <xs:attribute name="console" type="idString_t" use="optional" />
</xs:complexType>
<!-- Rsw -->
<xs:complexType name="rsw_e">
    <xs:all>
        <xs:element name="PhysicalMemoryAreas" type="memoryArea_e"/>
    </xs:all>
</xs:complexType>
<!-- Partition -->
<xs:complexType name="partition_e">
    <xs:all>
        <xs:element name="PhysicalMemoryAreas" type="memoryArea_e"/>
        <xs:element name="TemporalRequirements" minOccurs="0">
            <xs:complexType>
                <xs:attribute name="period" type="timeUnit_t" use="required"/>
                <xs:attribute name="duration" type="timeUnit_t" use="required"/>
            </xs:complexType>
        </xs:element>
        <xs:element name="HealthMonitor" type="healthMonitor_e" minOccurs="0" />
        <xs:element name="HwResources" type="hwResources_e" minOccurs="0" />
        <xs:element name="PortTable" type="partitionPorts_e" minOccurs="0" />
    </xs:all>
    <xs:attribute name="id" type="id_t" use="required"/>
    <xs:attribute name="name" type="idString_t" use="optional" />
    <xs:attribute name="console" type="idString_t" use="optional" />
    <xs:attribute name="flags" type="partitionFlagsList_t" use="optional" default="none" />
</xs:complexType>
<!-- Communication Ports -->
<xs:complexType name="partitionPorts_e">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Port">
            <xs:complexType>
                <xs:attribute name="name" type="idString_t" use="required"/>
                <xs:attribute name="direction" type="direction_t" use="required"/>
                <xs:attribute name="type" type="portType_t" use="required"/>
                <xs:attribute name="discipline" type="discipline_t" use="optional" />
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<!-- Channels -->
<xs:complexType name="channels_e">
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
            <xs:element name="Ipvi">
                <xs:complexType>
                    <xs:attribute name="id" type="id_t" use="required"/>
                    <xs:attribute name="sourceId" type="id_t" use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:sequence>
</xs:complexType>

```

```

        <xs:attribute name="destinationId" type="idList_t" use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="SamplingChannel">
      <xs:complexType>
        <xs:sequence minOccurs="1">
          <xs:choice>
            <xs:element name="Source" type="ipcPort_e" />
          </xs:choice>
          <xs:sequence minOccurs="1" maxOccurs="unbounded">
            <xs:choice>
              <xs:element name="Destination" type="ipcPort_e"/>
            </xs:choice>
          </xs:sequence>
        </xs:sequence>
        <xs:attribute name="maxMessageLength" type="sizeUnit_t" use="required"/>
        <xs:attribute name="refreshPeriod" type="timeUnit_t" use="optional" default="0s"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="QueuingChannel">
      <xs:complexType>
        <xs:sequence minOccurs="1">
          <xs:choice>
            <xs:element name="Source" type="ipcPort_e" />
          </xs:choice>
          <xs:choice>
            <xs:element name="Destination" type="ipcPort_e" />
          </xs:choice>
          <xs:sequence>
            <xs:attribute name="maxMessageLength" type="sizeUnit_t" use="required"/>
            <xs:attribute name="maxNoMessages" type="xs:positiveInteger" use="required"/>
            <xs:attribute name="maxTimeExpiration" type="timeUnit_t" use="optional" default="0s"/>
          </xs:sequence>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Devices -->
<xs:complexType name="devices_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="MemoryBlock" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="required"/>
          <xs:attribute name="start" type="hex_t" use="required"/>
          <xs:attribute name="size" type="sizeUnit_t" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Uart" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="required"/>
          <xs:attribute name="id" type="idString_t" use="required"/>
          <xs:attribute name="baudRate" type="xs:positiveInteger" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Null" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="idString_t" use="optional" />
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<!-- IPC Port -->
<xs:complexType name="ipcPort_e">
  <xs:attribute name="partitionId" type="id_t" use="required"/>
  <xs:attribute name="partitionName" type="idString_t" use="optional" />
  <xs:attribute name="portName" type="idString_t" use="required"/>
</xs:complexType>
<!-- Hw Description -->
<xs:complexType name="hwDescription_e">
  <xs:sequence>
    <xs:element name="MemoryLayout" type="memoryLayout_e"/>
    <xs:element name="ProcessorTable">
      <xs:complexType>
        <xs:sequence minOccurs="1" maxOccurs="256">
          <xs:element name="Processor" type="processor_e" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Devices" type="devices_e"/>
  </xs:sequence>
</xs:complexType>
<!-- Processor -->
<xs:complexType name="processor_e">
  <xs:all>
    <xs:element name="CyclicPlanTable" type="cyclicPlan_e"/>
  </xs:all>
  <xs:attribute name="id" type="id_t" use="required"/>
  <xs:attribute name="frequency" type="freqUnit_t" use="optional" />
  <xs:attribute name="features" type="processorFeaturesList_t" use="optional" default="none"/>
  <xs:attribute name="console" type="idString_t" use="optional" />
</xs:complexType>
<!-- HwResource -->
<xs:complexType name="hwResources_e">
  <xs:all>
    <xs:element name="IoPorts" type="ioPorts_e" minOccurs="0" />
    <xs:element name="Interrupts" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="lines" type="hwIrqIdList_t" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:all>

```

```

    <xs:element name="APBDev" type="APBDeviceType" minOccurs="0"/>
  </xs:all>
</xs:complexType>
<!-- Io Ports -->
<xs:complexType name="ioPorts_e">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="Range">
        <xs:complexType>
          <xs:attribute name="base" type="hex_t" use="required"/>
          <xs:attribute name="noPorts" type="xs:positiveInteger" use="required"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="Restricted">
        <xs:complexType>
          <xs:attribute name="address" type="hex_t" use="required"/>
          <xs:attribute name="mask" type="hex_t" use="optional" default="0x0"/>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
<!-- APBId -->
<xs:simpleType name="APBDevice_e">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:complexType name="APBDeviceType">
  <xs:attribute name="device" type="APBDevice_e"/>
</xs:complexType>
<!-- CyclicPlan -->
<xs:complexType name="cyclicPlan_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Plan" type="plan_e" />
  </xs:sequence>
</xs:complexType>
<!-- Plan -->
<xs:complexType name="plan_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Slot">
      <xs:complexType>
        <xs:attribute name="id" type="id_t" use="required"/>
        <xs:attribute name="start" type="timeUnit_t" use="required"/>
        <xs:attribute name="duration" type="timeUnit_t" use="required"/>
        <xs:attribute name="partitionId" type="id_t" use="required"/>
        <xs:attribute name="vCpuId" type="id_t" use="optional" default="0"/>
        <xs:attribute name="flags" type="slotFlagsList_t" use="optional"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="idString_t" use="optional"/>
  <xs:attribute name="id" type="id_t" use="required"/>
  <xs:attribute name="majorFrame" type="timeUnit_t" use="required"/>
</xs:complexType>
<!-- Health Monitor -->
<xs:complexType name="healthMonitor_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Event">
      <xs:complexType>
        <xs:attribute name="name" type="hmString_t" use="required"/>
        <xs:attribute name="action" type="hmAction_t" use="required"/>
        <xs:attribute name="log" type="yntf_t" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Memory Layout -->
<xs:complexType name="memoryLayout_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Region">
      <xs:complexType>
        <xs:attribute name="type" type="memRegion_t" use="required"/>
        <xs:attribute name="start" type="hex_t" use="required"/>
        <xs:attribute name="size" type="sizeUnit_t" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Hypervisor Memory Area -->
<xs:complexType name="hypMemoryArea_e">
  <xs:attribute name="size" type="sizeUnit_t" use="required"/>
  <xs:attribute name="flags" type="memAreaFlagsList_t" use="optional"/>
</xs:complexType>
<!-- Memory Area -->
<xs:complexType name="memoryArea_e">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="Area">
      <xs:complexType>
        <xs:attribute name="name" type="idString_t" use="optional" />
        <xs:attribute name="start" type="hex_t" use="required"/>
        <xs:attribute name="size" type="sizeUnit_t" use="required"/>
        <xs:attribute name="flags" type="memAreaFlagsList_t" use="optional" default="none"/>
        <xs:attribute name="mappedAt" type="hex_t" use="optional"/> <!-- default="" -->
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!-- Root Element -->
<xs:element name="SystemDescription">
  <xs:complexType>
    <xs:all>
      <xs:element name="HwDescription" type="hwDescription_e" />
      <xs:element name="XMHypervisor" type="hypervisor_e"/>
    </xs:all>
  </xs:complexType>
</xs:element>

```

```
<xs:element name="ResidentSw" type="rsw_e" minOccurs="0"/>
<xs:element name="PartitionTable">
  <xs:complexType>
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="Partition" type="partition_e" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Channels" type="channels_e" minOccurs="0" />
</xs:all>
<xs:attribute name="version" type="version_t" use="required"/>
<xs:attribute name="name" type="idString_t" use="required"/>
</xs:complexType>
</xs:element>
<!-- End Root Element -->
<!-- Elements -->
</xs:schema>
```