Muen System Specification

Adrian-Ken Rueegsegger, Reto Buerki

v0.7.2, April 9, 2024

Contents

List of Figures

Listings

Chapter 1

Introduction

The Muen system policy is a description of a component-based system running on top of the Muen Separation Kernel (SK). It defines what hardware resources are present, how many active components (called subjects) the system is composed of, how they interact and which system resources they are allowed to access. The contents a Muen system policy is composed of are outlined in chapter 2.

A system integrator specifies and configures such a component-based system at integration time in XML format. The Muen toolchain transforms the system description in multiple steps to the final system description, resolving abstractions which exist to make life simpler and less error-prone to the integrator. Additionally, the toolchain also creates various build artifacts which are incorporated into the system image. Chapter 3 gives an overview of the system integration process.

The Muen SK can be regarded as a policy enforcement engine, in the sense that it has no knowledge about the actual content of the generated data structures and in consequence the policy. For example, it knows nothing about the contents of subject page tables which define a subject's address space, nor does it know anything about its own page tables. In fact, these structures are not even mapped into the kernel.

The most important and final step in the integration of a Muen system is the actual generation of the data structures which guarantee subject isolation and the composition of the final system image. This step is performed by a trusted system composer called (static) $\tau 0$ (Tau Zero). The concept of $\tau 0$ is introduced in chapter 4.

Section 5 explains every tool and the system image composer in detail. It also presents the usage of each tool. Section 6 then outlines all semantic checks performed on the system policy primarily by the validation tool, but also by other tools in the toolchain.

Finally section 7 specifies the XML schema and structure of the source format of the Muen system policy. Explanations and examples illustrate how to configure a component-based system with the Muen SK.

Chapter 2

System Policy

The Muen policy specifies the following properties of a system:

- Configuration values
- Hardware resources
- Platform description
- Physical memory regions
- Device domains
- Events
- Communication channels
- Components
- Subjects
- Scheduling plans

The policy serves as a static description of a Muen system. Since all aspects of the system are fixed at integration time the policy is very well suited for automated as well as manual validation prior to system execution.

The details of each property above is outlined with examples in the XSD-schema of the format source policy in section 7.

2.1 Policy Format

The system policy is specified in XML. There are currently three different main policy formats:

- Source Format
- Format A
- Format B

The motivation to have several policy formats is to provide abstractions and a compact way for users to specify a system in format source while simultaneously facilitate traceability as well as reduced complexity of tools operating on the policy formats A and B.

The implementation of such tools is simplified by the absence of higher-level abstractions in the latter formats which would make the extraction of input data more involved.

Furthermore, the final format B must specify every aspect of the system explicitly, e.g. all attributes have a concrete value assigned, something which would be very tedious and repetitive and that burden should not be put on an integrator.

The following sections give more detail about each policy format.

☞ Only the policy in format source intended for system integrators is specified in this document. Other formats are processed by the toolchain and thus considered *internal*. While it is possible to specify a system policy in format A or B, it is not recommended.

Additionally to these three main formats, there may be extended versions of these formats if plugins are used. See section 5.2 for details about the plugin system.

2.1.1 Source Format

The user-specified policy is written in the so called *source format*. Constructs such as channels provide abstractions to simplify the specification of component-based systems. Many XML elements and attributes are optional and are *expanded* during later steps of the policy compilation process.

Kernel and $\tau 0$ subject (4.2) resources are not part of the source format since they are automatically added as part of the policy expansion step.

The use of configuration values enables parametrization of the system policy.

The policy in source format is specified in detail in section 7, while appendix 8.1 provides an annotated example policy illustrating the various policy elements.

2.1.2 Format A

Format A is a processed version of the source format where all inclusions of external files are resolved and abstractions such as channels have been deconstructed into their constituent parts. For example, a channel is expanded to a physical memory region and the corresponding writer and reader subject mappings with the appropriate access rights. Optional associated events have been automatically created and correctly linked with the designated subjects.

In this format all implicit elements, such as for example automatically generated page table memory regions, are specified. The kernel and $\tau 0$ configuration is also declared as part of format A.

The only optional attributes are addresses of physical memory regions.

2.1.3 Format B

Format B is equivalent to Format A except that all physical memory regions have a fixed location (i.e. their physical address is set).

Chapter 3

System Integration

A Muen system defined via the system policy is transformed and integrated by various tools to generate a bootable system image.

The directed graph 3.1 on page 9 illustrates the process.

At the top, the graph shows how configuration and build parameters are applied to the following constituents of the system policy:

- Hardware description (static)
  Contains manually specified devices by the integrator, e.g. common hardware like I/O ports of a PC speaker. Such devices are not automatically collected by the hardware configuration generator.
- Hardware description (generated)
  Hardware description extracted from a running Linux system by the `mugenhwcfg` tool (section 5.5.3).
- Platform description
  Common names and abstractions to form a unifying view over different hardware configurations. Additionally, platform-specific configuration values can be provided here.
- System description
  Specification of an actual component-based system running on the Muen SK.

These combined inputs form the parameterized system policy in format source, which can be used by components to extract system information. Such information might be for example the log channel count of a debug server subject, or whether a specific debug facility has been enabled by the system integrator.

The CSPECs mechanism outlined in the *Muen Component Specification* document [1] can be used by components to generate source specifications (e.g. in SPARK/Ada) from the component description. Furthermore, a component might expand its own component description with information extracted from system information, or it might use the `mucbinsplit` tool (section 5.5.4) to automatically fill in the memory regions provided by its binary after compilation. The expanded component description is then merged with the system policy for further processing.

After all component descriptions have been merged into the system policy, it is expanded by the expander tool (section 5.3.5). This step transforms the system policy from format source to format A. Abstractions like directed channels are now resolved to basic shared memory mechanisms and events, non-present optional attributes are added and set to default values.

The allocator tool (5.3.6) then loops over all physical memory regions which have no address assigned and places them in memory by allocating a region and thus a physical start address from the usable pool. The usable pool information is extracted from the allocatable memory block list (7.1.17) in the system policy. This process transforms the policy to format B where all elements must be present and attributes specified.

The policy is then checked for consistency and configuration errors by the validator tool (5.3.7). If a misconfiguration is found, the user is informed and the build aborts. The extensive checks performed by the validator tool are listed in section 6. If no error is found, the system policy is then ready to be used for three subsequent steps:

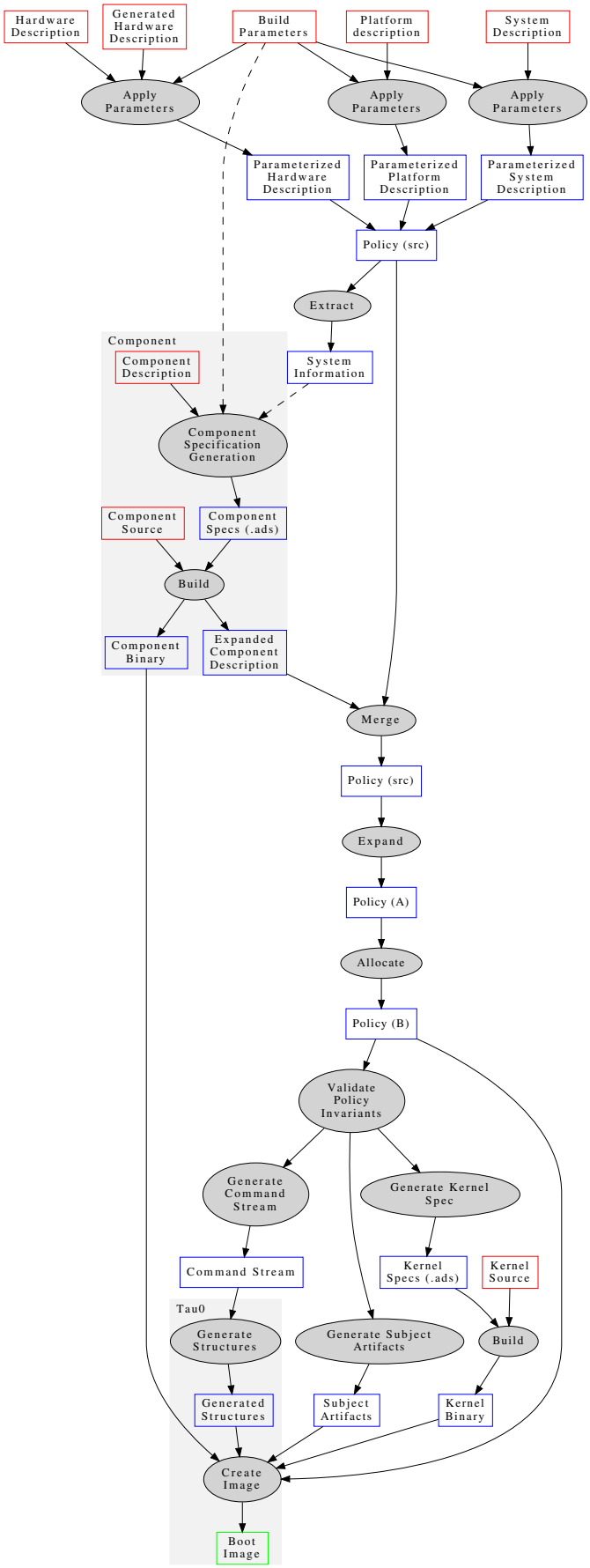


Figure 3.1: System Integration

- Generate kernel specifications (SPARK/Ada source files)
- Generate structures for subjects
- Create a command stream for $\tau 0$

The kernel source specifications contain tables and constants which represent the policy that is compiled into the kernel as part of the kernel build process and enforced at runtime.

An example of generated subject structures are ACPI tables, which are mapped into a Linux VM to announce the available hardware resources.

The command stream generator (5.3.9) generates instructions in XML format for the $\tau 0$ system composer explained in the following section.

Chapter 4

Tau0 Concept

The XML command stream together with the other build artifacts like subject structures or the kernel binary is provided as input to the trusted system composer $\tau 0$. Its task is to compose a system image while making sure that certain invariants are not violated. The $\tau 0$ concept is a mechanism to gradually increase the flexibility of a component based system while keeping a high level of assurance regarding the correctness of isolation enforcement.

There are two modes of operation for $\tau 0$:

- static
- dynamic

In the static scenario, the task of $\tau 0$ is to construct a bootable system image by assembling the input files and generating data structures such as page tables, all while checking that invariants necessary for correct isolation are valid. An example for such an invariant is that no subject memory mapping may reference a memory region containing paging structures.

For the dynamic case, the goal is to have a trusted $\tau 0$ subject with additional privileges to interact with the Muen SK over a special $\tau 0$ interface. This will allow $\tau 0$ to change certain clearly defined aspects of the system state at runtime. A potential use-case would be to set up a new subject, assign resources like memory and devices to it and then instruct the kernel to schedule it.

Since it must be guaranteed that a dynamic system is as secure as the static one, $\tau 0$ must be developed with the same care as the kernel itself, meaning it must be written in SPARK/Ada and security properties which provide hard isolation must be formally proven. This process is currently ongoing.

4.1 Static Mode of Operation

The static variant is the one which is currently implemented. $\tau 0$ runs at integration time and assembles the system image by constructing the data structures guaranteeing isolation and merging in the build artifacts of the other Muen tools, like untrusted ACPI data structures for Linux VMs.

Static $\tau 0$ fulfills its task by creating the system image in memory while processing the commands from the command stream. See listing 5.1 on page 20 for an example command stream.

$\tau 0$ is written in SPARK/Ada and it applies memory typization to formally prove aspects of the system. Command processing starts from a well-known good state and it is enforced that each system state transition resulting from a new command input results in a good state again by showing that invariants hold after the transition. If not, the command is rejected and the build aborts.

See the project README or the webpage[1] for more information about the current state of $\tau 0$.

4.2 Dynamic

While the system image is composed by the static variant of $\tau 0$, the goal is to run the same code as $\tau 0$ subject at runtime. Note that this is not yet implemented but planned as a way

[1] https://muen.sk/tau0.html

forward to achieve more dynamic systems while having the same assurance about security and safety properties.

The dynamic $\tau 0$ running as subject will reconstruct the system state defined at integration time and continue to process commands starting from there. Depending on the system use case, commands might be sent to dynamic $\tau 0$ by a special control subject.

The dynamic variant can be divided into multiple sub-variants, depending on how much dynamic system behavior is allowed. For example, the initial dynamic variant might only allow entity construction, not destruction.

Chapter 5

Toolchain

5.1 Overview

While the previous section 3 presented an overview of the system integration process and section 4 introduced the $\tau 0$ concept, this section focuses on the detailed description of the tools forming the Muen toolchain.

The tool-based processing of the Muen system policy can be divided into the following steps:

- Policy merging
- Components build
- Components specification merging
- Policy compilation
- Policy validation
- Structure generation
- Command stream generation for $\tau 0$
- Image generation by $\tau 0$

Following the Unix philosophy "A program should do only one thing and do it well" each of the tools only performs a specific task. They work in conjunction to process a user-defined policy and build a bootable system image. Figure 5.1 presents another illustration of the policy processing, this time laying the focus on the tools. The following sections explain each processing step while section 5.3 describes each tool separately.

5.1.1 Policy Merging

The Merger tool outlined in section 5.3.3 is responsible to merge XML files stored at different locations on the file system into one system policy in format source.

The tool reads a system configuration in XML format to locate the following files:

- System policy
- Hardware specification
- Additional hardware specification
- Platform specification

To make the system description flexible and modular the following features are supported in the input policy:

- The tool provides an implementation of the XML XInclude mechanism[1]. Using includes, the policy writer is able to separate and organize the system policy as desired. Instead of specifying the whole policy in one file, the subject specifications can be split into separate files, or common parts shared by different system descriptions can be extracted.
- Expressions can be used to formulate (nested) terms using equality/inequality, numeric and logical operators as well as concatenation of strings. Expressions can be used just like configuration variables to provide parameters for other mechanisms.
- The use of conditionals enables selective activation of parts of the source policy depending on the value of a given configuration variable. This allows flexible customization of a system during policy compilation by setting the value of a configuration variable or formulating an appropriate expression.
- Configuration variable substitution enables the policy writer to set the value of attributes to those of referenced configuration variables or expressions. Attributes that start with a dollar sign followed by a variable name are substituted by the value of the variable.
- The policy may define templates for XML code, including parameters that can be used within expressions, conditionals and references within that template. Templates can be employed to avoid code duplication and to encapsulate portions of code. Hence, templates can define building blocks and help to provide a high-level view of a system.
- When two building blocks are connected via a channel or when a subject behaves like a client of another subject, it is desirable to insert a communication channel into a subject from "outside" of that subject. Such additions to an XML node are possible with `amend` statements. On evaluation, the children of an `amend`-node are merged into the children of the node specified by the given XPath.

After the merge step, the resulting policy is well formatted to minimize the difference in the generated policies resulting from the subsequent tasks. This allows the user to easily review (`diff`) and therefore verify the results of each policy compilation task.

5.1.2 Components Build

After hardware, platform and high-level system policy are merged into a single source policy file, components may extract relevant information. For example an XSL transformation (XSLT) script could extract the I/O port of a specific device and create a corresponding configuration value based on it, which is then included in the component specification.

The `mucfgcvresalloc` tool described in 5.3.1 implements the blue *Component VR Allocator* task shown in figure 5.1. It is primarily used to automatically allocate attributes of component resources like *virtualAddress* for channels. Furthermore, similar to the policy merger, it supports inclusions, conditionals, expressions and configuration value substitutions.

After the component specification has been processed, the `mucgenspec` tool described in 5.3.11 generates Ada/SPARK packages containing constants derived from the declared component resources and config values. These constants can be used to reliably address specific or configurable resources in the source code. With these constants the component source code is compiled into a binary.

The `mucbinsplit` tool described in 5.5.4 can be used to extract ELF sections of the component binary into separate files. It automatically extends the component specification by adding a corresponding memory region with the appropriate access rights (e.g. executable, writable) for text, rodata, data, bss and stack sections.

5.1.3 Components Specification Merging

The processed component specifications are merged into the system source policy by the Muen component specification joiner tool described in section 5.3.4.

This step is optional as static component specifications which need no processing can also be manually specified in the system policy directly.

[1] http://www.w3.org/TR/xinclude-11/

5.1.4 Policy Compilation

Policy compilation encompasses the tasks involved to transform the policy from source format to format A and finally to format B, which is the fully expanded format with no implicit properties.

The Virtual Resource Allocator tool sets virtual resources that are left unassigned in the joined policy. These resources include virtual addresses, event numbers and vectors of channels. Often, the precise values of these resources do not matter to the integrator and make the policy harder to read and write. Section 5.3.2 explains the Virtual Resource Allocator tool in detail.

The Expander tool takes care of completing the user-specified policy with additional information and resolving abstractions only available in format source to their corresponding low-level constructs.

For example, the concept of *channels* only exists in format source. Therefore a channel specified in format source must be expanded to shared memory regions with optional associated events in format A. Also, the Expander tool inserts specifications for the Muen kernel itself so the user is lifted from that burden. Generally, the aim of the expansion task is to make the life of a policy writer as easy as possible by expanding all information which can be derived automatically. Section 5.3.5 explains the Expander tool in detail.

The result of the expansion task is a policy in format A which is the input for the Allocator tool. This tool is responsible to assign physical memory addresses to all memory regions which are not already explicitly placed in memory. By querying the hardware section of the policy, the tool is aware of the total amount of available RAM on a specific system and allocates regions of it for memory elements with no explicit physical address. The Allocator tool also implements optimization strategies to keep the resulting system image as small as possible. For example, file-backed memory regions (e.g. a memory region storing a component executable) are preferably placed in lower physical regions. See section 5.3.6 for a description of the Allocator tool.

After the allocation task is complete, the policy is stored in format B. This format states all system properties explicitly and is used as input for the Validation step.

5.1.5 Policy Validation

Before structures required to pack the final system image are generated, the policy must be thoroughly validated to catch errors in the system specification. Such errors might range from overlapping memory, undefined resource references to incomplete scheduling plans etc. The Validator task performs checks that assure the policy in format B is sound and free from higher-level errors that are not covered by XML schemata restrictions.

It is important to always run the Validator as the system could otherwise exhibit unexpected behavior. This is especially true if a policy writer decides to specify the system directly in format B which is also possible but not advised. Section 5.3.7 explains the usage of the Validator tool, while section 6 outlines all performed checks.

It should be noted that correct memory typization and all invariants enforced by $\tau 0$ when constructing the system image cannot be bypassed, since the checks are inherent to the generation of the bootable image file.

5.1.6 Structure Generation

The structure generation step encompasses various tools which extract information from a policy in format B and generate files in different formats.

While some generated files are directly linked into the Muen kernel (i.e. Source Specifications, see 5.3.11), most of them are subject-related. Depending on the subjects included in the actual system policy, the following subject structures are generated:

- MSR store regions
- Sinfo regions
- Regions for Linux VMs
  - ACPI tables
  - Linux zero-page (ZP) regions

- Regions for MirageOS/Solo5[2] unikernels
  - Solo5 boot info

As these structures do not affect isolation between subjects or subjects and the kernel, they are not generated by $\tau 0$ but only included as binary data via XML command stream and build artifacts.

The structure generator tools are explained in section 5.3.11.

5.1.7 Image Creation

The system image composer assembles the final system image. This task is performed by $\tau 0$ static introduced in the previous section 4.1. The usage of it is specified in 5.3.10.

5.2 Plugin System

As is shown in figure 5.1, the build process includes two steps where plugins may be inserted if needed. The intention of the plugin system is to simplify the introduction of small, less critical modifications to the toolchain and keep those separate from the "core"-toolchain of Muen. A tool is considered to be part of the core-toolchain if it is *needed* to build the bootable system image. Additionally, `Mucfgvalidate` is considered part of the core-toolchain, too.

An example of a plugin is the documentation-plugin (see 5.4.2). It can help to build documentation for a system and the user can easily adjust its functionality and the underlying XSD-schema as needed. Similarly, a plugin can be used to extend the automatically generated headers of components.

To hide the additional information in the policy from tools that may not be able to digest them, `muxmlfilter` (see 5.4.1) is used to remove such parts of the policy before processing it. That way, changes in the plugins have minimal impact on the core toolchain.

To enable the validation of extended policies, the plugin system includes tools that extend policy format definitions. Hence, there may be extended versions of Source Format, Format A and Format B, depending on plugin usage.

5.3 Core Tools

This section describes the tools which form the core of the Muen toolchain.

5.3.1 Component Virtual Resource Allocator

The tool `mucfgcvresalloc` processes a user provided component specification and outputs a finished, schema compliant description of the component interface. Just like `mucfgmerge` it supports inclusion of external files, conditionals, expressions and substitutions. Furthermore, it can automatically allocate virtual resources as described below.

Name
`mucfgcvresalloc`

Input
Component configuration as XML, colon-separated list of include paths

Output
Component specification as XML in component format

The main processing steps are:

1. Merge XIncludes of main XML file, i.e., insert the referenced files at the given location;
2. Evaluate expressions, conditionals and substitutions;
3. Assign missing virtual addresses of channels and memory regions, as well as arrays of channels and memory regions;

[2] https://github.com/Solo5/solo5

4. Assign missing IRQs of channel reader events and arrays of channel readers;

5. Assign missing event IDs of channel writer events and arrays of channel writers;

Virtual addresses, IRQs and event IDs are considered to be *virtual resources*. Each of the three virtual resources has its own domain. To request automatic allocation of the attribute `virtualAddress` (or `virtualAddressBase` for arrays) omit the attribute. To request automatic allocation of event ids of readers or writers the respective attribute must be set to `auto`.

To determine which addresses or ids to choose, `mucfgcvresalloc` looks at the addresses and ids already present in the file. Dependencies of virtual resources through libraries or the system policy can be resolved by manually assigning such resources.

5.3.2 Virtual Resource Allocator

The tool `mucfgvresalloc` processes the joined system policy and outputs a system policy in format source. Its purpose is to automatically allocate virtual resources within subject descriptions.

Name
`mucfgvresalloc`
Input
Joined system policy as XML
Output
System policy where all virtual resources of subjects have been assigned

`mucfgvresalloc` can assign virtual addresses of channels and memory regions, IRQs of channel reader events, and IDs of channel writer events. These attributes are considered to be *virtual resources*. Each of the three virtual resources has its own domain. To request automatic allocation of the attribute `virtualAddress` omit the attribute. For the other two resources the respective attribute must be set to `auto`.

To determine which value to choose for a virtual resource, `mucfgvresalloc` processes each subject separately. For each subject, it gathers virtual resources already set in the component that is referenced by the subject and resources set in the subject itself. Thereafter, missing virtual resources are either set to the value fixed by the component (if possible) or chosen from the remaining space in the respective domain.

Dependencies of virtual resources through libraries or devices can be resolved by manually assigning such resources.

5.3.3 Policy Merger

The merger tool `mucfgmerge` combines user-provided system policy files into a single XML document.

Name
`mucfgmerge`
Input
System configuration as XML, colon-separated list of include paths
Output
System policy in format source (merged)

This tool reads the system configuration and merges the specified system policy, hardware and platform files into a single file. To ease the creation of many similar variants of a system the provided system policy may use templates, expressions, conditionals and amend statements. The main processing-steps are:

1. Merge XIncludes of system policy into the system policy, i.e., insert the referenced files at the given location;

2. Merge hardware, additional hardware and platform specifications into the system policy. This includes merging the platform configuration section into the global configuration section;

3. Instantiate the templates, using the provided values and variable names;

4. Evaluate expressions, resulting in new configuration variables;
5. Replace all references to configuration variables with their value;
6. Evaluate conditionals, i.e., decide which sub-trees of the XML-tree to discard;
7. Evaluate amend statements, i.e., move sub-trees within the XML-tree.

The result is in policy source format and re-formatted so changes to the policy by subsequent build steps can be manually reviewed or visualized by diffing the files. In particular, the result does not contain any templates, expressions, conditionals or amend statements. The tool has debug modes that increase the verbosity of the output, in particular in case of errors.

5.3.4 Component Specification Joiner

The Muen component specification joiner adds component XML specifications to the component section of a specified system policy and writes the result to a designated output file. Each given component/library specification is loaded and validated against the component specification XML schema. If it is correct the content is added to the components section of the system policy specified as input file. If the given system policy does not yet contain a components section, it is created. The result is written to the file specified by the `-o` parameter. In-place processing is supported by passing in the same value for input and output file.

Name
`mucfgcjoin`
Input
System policy in format source, comma-separated list of component specs
Output
System policy in format source (joined)

5.3.5 Expander

The expander completes the user-provided system policy by creating or deriving additional configuration elements.

Name
`mucfgexpand`
Input
System policy in format source
Output
System policy in format A (expanded)

The Expander performs the following actions:

- Pre-check the system policy to make sure it is sound
- Expand channels
- Expand device resources
- Expand device isolation domains
- Expand kernel sections
- Expand minimal $\tau 0$ subject
- Expand additional memory regions
- Expand hardware-/platform-related information
- Expand additional subject information
- Expand profile-specific information
- Expand scheduling information
- Post-check resulting policy

5.3.6 Allocator

The Allocator is responsible to assign a physical address to all global memory regions.

Name
`mucfgalloc`
Input
System policy in format A
Output
System policy in format B (allocated)

First, the Allocator initializes the physical memory view of the system based on the physical memory blocks specified in the XML hardware section. It then reserves memory that is occupied by pre-allocated memory elements (i.e. memory regions with a physical address or device memory). Finally it places all remaining memory regions in physical memory. In order to reduce the size of the final system image file-backed memory regions are placed at the start of memory.

5.3.7 Validator

The Validator performs additional checks that go beyond the basic restrictions imposed by the XML schema validation. For example it checks that the hardware provides an IOMMU device and that all references to subjects are resolvable. See 6 for a complete list of all executed checks. The tool aborts with a non-zero exit status and an explanatory message to the user if checks fail.

Name
`mucfgvalidate`
Input
System policy in format B
Output
None, raises exception on error

5.3.8 Hasher

The `mucfgmemhashes` tool is used to add memory integrity hashes to a given policy.

Name
`mucfgmemhashes`
Input
System policy in format B, colon-separated list of input directories containing build artifacts
Output
System policy in format B with memory integrity hashes

The tool appends a hash to all memory regions with fill and file content. It must run after all files have been generated by the structure generator tools.

The actual hash is generated using the SHA-256 algorithm and is intended to be used to verify the integrity of memory regions during runtime.

Note that no hashes are generated for sinfo memory regions. Since the hash information is exported via sinfo, and the sinfo region is itself part of the memory information of a subject, this hash would be self-referential.

The tool also replaces all occurrences of `hashRef` elements. A hash reference element instructs the tool to copy the hash element of the referenced memory region after message digest generation.

From an abstract point of view, the `hashRef` element is a way to link multiple memory regions by declaring that the hash of the content is the same. The hash may serve as an indicator on how to reconstruct the (initial) content of a memory region. This mechanism is used by e.g. the subject loader (SL) during subject init and reset operation. The subject loader expansion step remaps writable memory regions of the loadee (the subject under loader control) to SL and replaces the original regions with new ones containing a hash reference to the associated physical memory region. This way SL is able to determine the intended content of the target memory region by looking up the region in its sinfo page using the hash value as key.

5.3.9 Tau0 Command Stream Generator

The `mugentau0cmds` tool creates an XML command stream for $\tau 0$ to let it compose the system image specified by the system policy given as input.

The tool reads the policy in format B and translates it to a sequence of commands as shown in the following listing:

```
<tau0>
 <commands>
  <addMemoryBlock address="16#0000#" size="157"/>
  <addMemoryBlock address="16#0010_0000#" size="130816"/>
  <addMemoryBlock address="16#2020_0000#" size="130564"/>
  <addMemoryBlock address="16#4000_5000#" size="453932"/>
  <addMemoryBlock address="16#bae9_f000#" size="256"/>
  <addMemoryBlock address="16#baf9_f000#" size="96"/>
  <addMemoryBlock address="16#0001_0000_0000#" size="3401216"/>
  <addMemoryBlock address="16#ba3b_a000#" size="23"/>
  <addMemoryBlock address="16#bb80_0000#" size="16896"/>
  <addProcessor id="0" apicId="0"/>
  <addProcessor id="1" apicId="2"/>
  <addIoapic sid="16#f0f8#"/>
  <createLegacyDevice device="0"/>
  <addIOPortRangeDevice device="0" from="16#03c0#" to="16#03df#"/>
  <addMemoryDevice device="0" caching="WC" address="16#000a_0000#" size="32"/>
  <activateDevice device="0"/>
  <createLegacyDevice device="1"/>
  <addIOPortRangeDevice device="1" from="16#0060#" to="16#0060#"/>
  <addIOPortRangeDevice device="1" from="16#0064#" to="16#0064#"/>
  <addIRQDevice device="1" irq="1"/>
  <addIRQDevice device="1" irq="12"/>
  <activateDevice device="1"/>
  <createLegacyDevice device="2"/>
  <addIOPortRangeDevice device="2" from="16#0070#" to="16#0071#"/>
  <activateDevice device="2"/>
  <createLegacyDevice device="3"/>
  <addIOPortRangeDevice device="3" from="16#0cf9#" to="16#0cf9#"/>
  <addIOPortRangeDevice device="3" from="16#0404#" to="16#0404#"/>
  <activateDevice device="3"/>
  <createLegacyDevice device="4"/>
  <addMemoryDevice device="4" caching="UC" address="16#fec0_0000#" size="1"/>
  <activateDevice device="4"/>
```

Listing 5.1: $\tau 0$ Command Stream

As $\tau 0$ strictly enforces certain invariants, the system must be constructed in a way not to violate these invariants. For example, before memory can be typed as being a VT-d root table, this memory must be cleared. Otherwise the memory typing model of $\tau 0$ is violated.

The `mugentau0cmds` tool must take this into consideration when iterating over the resources specified in the input system policy and generating commands which instruct $\tau 0$ to create the specified system.

Name
`mugentau0cmds`

Input
System policy in format B

Output
XML command stream for $\tau 0$ static

5.3.10 Tau0 Static

The $\tau 0$ static component serves as an image composer during integration. The concept and motivation of this approach is described in chapter 4.

Name
`tau0_main`

Input
XML command stream, colon-separated list of input directories containing build artifacts

Output
Muen system image

Output format
Command Stream Loader (CSL) image[3], bootable by any compliant bootloader.

[3] https://www.codelabs.ch/download/bsbsc-spec.pdf

If a command is received which violates a constraint enforced by $\tau 0$ static, the tool aborts system image construction, displays an error message and returns with a non-zero exit status.

5.3.11 Structure Generators

These tools do not change the policy and use it read-only.

MSR Stores Generator

Generate MSR store for each subject with MSR access.

Name
`mugenmsrstore`
Input
System policy in format B
Output
MSR store files of subjects in binary format
Output format
Intel SDM Vol. 3C, "24.8.2 VM-Entry Controls for MSRs" and Intel SDM Vol. 3C, "24.7.2 VM-Exit Controls for MSRs".

The tool generates MSR stores for each subject. The MSR store is used to save/load MSR values of registers not implicitly handled by hardware on subject exit/resumption.

MSR stores are used by hardware (VT-x) to enforce isolation of MSR (i.e. subjects that have access to the same MSRs cannot transfer data via these registers).

ACPI Tables

Generate ACPI tables for all Linux subjects.

Name
`mugenacpi`
Input
System policy in format B
Output
ACPI tables of all Linux subjects
Output format
Advanced Configuration and Power Interface (ACPI) Specification[4]

ACPI tables are used to announce available hardware to VM subjects. A set of tables consists of an RSDP, XSDT, FADT and DSDT table. See the ACPI specification for more information about a specific table.

Linux Zero Pages

Generate Zero Pages for all Linux subjects.

Name
`mugenzp`
Input
System policy in format B
Output
Zero pages of all Linux subjects
Output format
Linux Boot Protocol[5]
Zero Page[6]

[4] http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf
[5] https://www.kernel.org/doc/Documentation/x86/boot.txt
[6] https://www.kernel.org/doc/Documentation/x86/zero-page.txt

The so-called Zero Page (ZP) exports information required by the boot protocol of the Linux kernel on the x86 architecture. The kernel uses the provided information to retrieve settings about its runtime environment:

- Type of bootloader
- Map of physical memory (e820 map)
- Address and size of initial ramdisk(s)
- Kernel command line parameters

Solo5 Boot Info

Generate Solo5 boot info structures for MirageOS unikernels[7] running on the Solo5 platform.

Name
`mugensolo5`
Input
System policy in format B
Output
Solo5 boot info for all MirageOS subjects
Output format
struct hvt_boot_info[8]

The boot info structure exports information required by Solo5. The unikernel uses the provided information to retrieve settings about its runtime environment:

- Memory size in bytes
- Address of end of unikernel
- CPU cycle counter frequency, Hz
- Address of command line (C string)
- Address of application manifest

Kernel Source Specifications

Generate source specifications used by kernel.

Name
`mugenspec`
Input
System policy in format B
Output
Source specifications in SPARK, C and GPR format

Gathers data from the system policy to generate various source files in SPARK, C and GNAT project file (GPR) format. Created output includes constant values for memory addresses, device resources, scheduling plans, etc. See the description of the `Skp` package hierarchy in the Muen Kernel Specification document [2] for the exact information these packages provide.

[7] https://mirage.io
[8] https://github.com/Solo5/solo5/blob/master/include/solo5/hvt_abi.h

Component Source Specifications

Process component description and generate source specifications from it.

Name
`mucgenspec`
Input
Component description in XML
Output
Component source specifications in SPARK

The component spec generation tool generates Ada/SPARK packages containing constants of the declared logical component resources. The generated specifications can be used in the component source code to access the declared resources.

Subject Info (sinfo)

Generate subject information data for each subject.

Name
`mugensinfo`
Input
System policy in format B
Output
Subject info data in binary format
Output format
As specified in [1] and `common/musinfo/musinfo.ads`

The Sinfo page is used to export subject information data extracted from the system policy to subjects. Currently, information about available memory regions, communication channels, events, vectors and assigned PCI devices is provided.

5.4 Plugins

This section lists tools which are either plugins for the toolchain or support the plugin system.

5.4.1 XML Filtering

The `muxmlfilter` tool can filter a given XML-file such that the output satisfies a given schema definition.

Name
`muxmlfilter`
Input
XSD-Schema, XML-file
Output
Filtered XML-file where all nodes not allowed by the given schema have been deleted

The filter is intended for XML files that satisfy the target schema, except for some added element-nodes that are not allowed by the schema. The possibility to filter out some elements is needed to make toolchain-plugins possible (see 5.2).

The given schema can either be one of the built-in schemata (in which case it can be given by name), or it can be specified by a file-path. While `muxmlfilter` is not specific to Muen policies, the used schema needs to satisfy a number of technical restrictions. These are described in `tools/libmuxml/schema_plugins/README.md`

5.4.2 Documentation Transfer

The `doc_transfer` script can extend a given policy B with documentation nodes.

Name
`doc_transfer`
Input
Policy B XML-file, joined policy XML-file
Output
Policy B with added documentation

The tool processes the given joined policy and extracts a list of target XPath-addresses paired with an XML-node containing documentation. This list is then applied to the given policy B, which effectively transfers documentation from the given joined policy to the given policy B. This way, documentation information that was present in policy src can be reinserted in policy B. The amended policy B can serve as a single source for documentation information.

5.5 Additional Tools

This section lists additional helper tools which simplify the process of generating and validating a Muen system.

5.5.1 Kernel ELF Checker

The `mucheckelf` tool enforces that the format of a given Muen kernel ELF binary matches the kernel memory layout specified in a system policy. Furthermore, the ELF kernel entry point is compared to the expected value.

Size, VMA (Virtual Memory Address) and permissions of binary ELF sections are validated against kernel memory regions defined in the policy. The following table lists the correspondence of ELF section names to logical kernel memory region names.

ELF Section	Memory Name
.text	kernel_text
.data	kernel_data
.rodata	kernel_ro
.bss	kernel_bss
.globaldata	kernel_global_data

5.5.2 Stack Usage Checker

The `muchecksstack` tool statically calculates the worst-case stack usage of a native Ada/SPARK component or the Muen kernel compiled with the -fcallgraph-info switch[9].

The tool takes a GNAT project file and a stack limit in bytes as input. All control-flow information (.ci) files found in the object directory of the main project and all of its dependencies are parsed. Once the control-flow graph is constructed the maximum stack usage of each subprogram is calculated and checked against the user-specified limit. The tool exits with a failure if a stack usage exceeding the limit is detected.

Note that the tool is not applicable to arbitrary software projects as it does not handle dynamic/unbounded stack usage and recursion. In the context of the Muen project these cases can not occur since they are prohibited by the following restriction pragmas:

- No_Recursion
- No_Secondary_Stack
- No_Implicit_Dynamic_Code

Additionally, the `-Wstack-usage` compiler switch warns about potential unbounded stack usage.

[9] https://www.adacore.com/uploads/technical-papers/Stack_Analysis.pdf

5.5.3 Hardware Config Generator

The `mugenhwcfg`[10] tool has been created to automate the process of gathering all necessary hardware information. To collect data for a new target hardware all that is required is to run the tool on a common Linux distribution[11]. See the project README for more information.

Name
`mugenhwcfg`
Input
None
Output
Hardware description in `output.xml`

The tool is implemented in a way to extract as much information from the system and generate a hardware configuration even if problems are encountered. The aim is to assist the integrator as much as possible in writing a hardware configuration for the target hardware.

Therefore, the tool only fails with a non-zero exit status and no output if essential required data can not be extracted from the system. Other problems are reported in the potentially incomplete `output.xml` file as XML comments, making the encountered problems on the actual machine evident. The following snippet provides an example of such a warning comment in the header of the generated `output.xml` file:

```
* WARNING *: Unable to resolve device class 0c80. Please update pci.ids
(-u) and try again
```

The comments should be rather self-explanatory. In this case, the problem is only a minor issue since the tool was simply unable to resolve a device class number to a human-readable string.

The next example has more consequences:

```
* WARNING *: Skipping invalid IRQ resource for device 0000:00:1f.3: None
```

This has the effect that no IRQ resource is appended in the specification of the device exhibiting this problem. While the device can still be assigned to a subject, it is missing the IRQ element and as a result the IRQ resource itself. It can be assumed that this leads to problems with the driver interacting with the device. For proper operation, it is the policy writer's task to rectify the hardware specification by determining the correct configuration manually.

5.5.4 Component Binary Splitter

The `mucbinsplit` tool splits component binaries into multiple files, one per ELF section.

Name
`mucbinsplit`
Input
Component description in XML, Component ELF binary
Output
Binary files corresponding to ELF sections, processed component description in XML

The component binary splitter tool processes component binaries and creates a separate file for each ELF section. The component XML description is extended by adding a file-backed memory region for each ELF section with the appropriate virtual mapping address, size and access rights. The RIP value is set to the ELF entry point of the component binary.

The resulting processed component description is written to the given output location while the binary section files are written to the specified output path.

[10] https://git.codelabs.ch/?p=muen/mugenhwcfg.git
[11] https://github.com/roburio/mugenhwcfg-live

5.5.5 Microcode Updates

The `mucfgucode` tool is used to enable Intel processor microcode updates (MCU) on Muen.

Name
: `mucfgucode`

Input
: System policy in format source, directory containing Intel microcode updates

Output
: System policy in format source containing a file-backed microcode memory region. The Intel microcode update blob is copied to the specified output directory

The microcode update tool processes a system policy in format source, extracts the signature of the target processor from the CPUID leaf 1 eax register value and executes the `iucode-tool`[12] for the specified directory containing Intel microcode updates. See the Intel SDM Vol. 3A, "9.11 Microcode Update Facilities" for more information on Intel MCU.

The tool adds a file-backed physical memory for a matching MCU of the given processor. Also, the MCU is copied to the specified output directory with a `.ucode` suffix.

If a physical memory region for MCU already exists, it is removed before further processing. The system policy is left unchanged if the tool does not find an applicable MCU for the target processor.

[12] https://gitlab.com/iucode-tool/iucode-tool/-/wikis/home

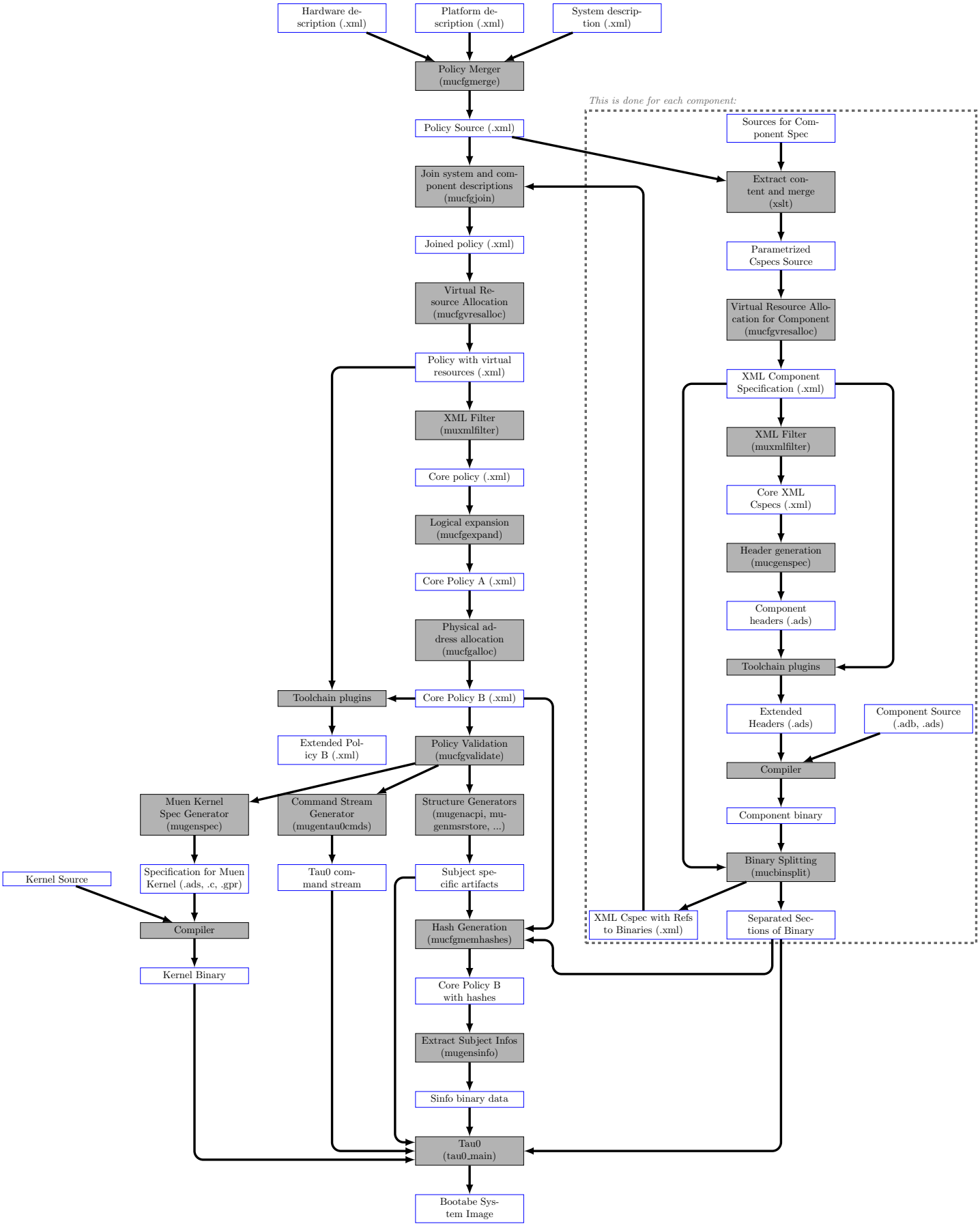


Figure 5.1: Build process

Chapter 6

Policy Validation

Prior to operate on the policy, any tool outlined in the toolchain section 5.3 checks all required preconditions by running *validator* procedures. For example a tool accessing physical devices via subject logical device references will execute a validator checking such references for validity.

Before the policy is used to generate system structures like sinfo regions, or the command stream for $\tau 0$, the expanded policy in format B is validated by executing a comprehensive set of checks. This is done by the `mucfgvalidate` tool outlined in section 5.3.7.

The following sections list the various checks executed by `mucfgvalidate` and the other Muen build tools in the toolchain.

6.1 Configuration

The following checks are performed to guarantee correctness of configuration options in the system policy.

- Validate config variable name uniqueness.
- Check that all booleans defined in config contain a valid value.
- Check that all integers defined in config contain a valid value.
- Check that all expression config variable references are valid.
- Check that all integers defined in expressions contain a valid value.
- Check that all booleans defined in expressions contain a valid value.
- Check that all conditional config variable references are valid.

6.2 Devices

The following checks are performed to guarantee that hardware devices are correctly configured in the system policy.

- Validate that devices referenced by logical devices exist.
- Validate that device names (including device aliases/classes) are unique.
- Validate that all physical IRQs are unique.
- Validate that physical device IRQs referenced by logical IRQs exist.
- Validate that ISA IRQs fulfill their constraints.
- Validate that PCI LSI IRQs fulfill their constraints.
- Validate that PCI MSI IRQs fulfill their constraints.
- Validate that PCI MSI IRQs are consecutive.

- Validate that physical IRQ names are unique per device.
- Validate that all I/O start ports are smaller than end ports.
- Validate that physical I/O ports referenced by logical I/O ports exist.
- Validate that all physical I/O ports are unique.
- Validate that physical I/O port names are unique per device.
- Validate that device memory names are unique per device.
- Validate that device memory referenced by logical device memory exists.
- Validate that PCI device bus, device, function triplets are unique.
- Validate that logical device references of each subject do not refer to the same physical device.
- Validate that PCI device reference bus, device, function triplets are unique per subject.
- Validate that all device references specifying a bus, device, function triplet are references to physical PCI devices.
- Validate that all device references to PCI multi-function devices belong to the same subject and have the same logical device number.
- Validate that all device references not specifying a bus, device, function triplet are references to physical legacy (non-PCI) devices.
- Validate that all logical PCI devices specify bus number zero.
- Validate that all IOMMU memory-mapped IO regions have a size of 4K.

6.3 Device Domains

The following checks are performed to guarantee that IOMMU device domains are correctly configured in the system policy.

- Validate that domain device references are unique.
- Validate that no virtual memory regions of a domain overlap.
- Validate that domain memory referenced by subjects is mapped at the same virtual address.
- Validate memory type of physical memory referenced by domains.
- Validate that each device referenced by a device domain is a PCI device.
- Validate that each device domain has a physical PT memory region.
- Validate that each PCI bus has a physical VT-d context memory region.

6.4 Events

The following checks are performed to guarantee that events are correctly configured in the system policy.

- Check that all physical event names are unique.
- Check that each global event has associated sources and one target.
- Check subject event references.
- Validate that there are no self-references in subject's event notification entries.
- Validate that notification entries switch to a subject running on the same core and in the same scheduling group.

- Validate that target subjects of IPI notification entries run on different logical CPUs.
- Validate that target event IDs as well as logical names are unique.
- Validate that source event IDs as well as logical names are unique per group.
- Check source event ID validity.
- Check that source event IDs of the VMX Exit group are all given or a default is specified.
- Check that self events provide a target action.
- Check that kernel-mode events have an action specified.
- Check that system-related actions are only used with kernel-mode events.
- Check that level-triggered IRQs have a corresponding unmask IRQ event.

6.5 Files

The following file-specific checks are performed.

- Check existence of files referenced in XML policy.
- Check if files fit into corresponding memory region.

6.6 Hardware

The following checks are performed on the hardware section of the policy.

- Validate that memory regions fit into available hardware memory.
- Validate that no memory blocks overlap.
- Validate that the size of memory blocks is a multiple of page size.
- Validate that PCI config space address and size are specified if PCI devices are present.
- Validate that the hardware provides enough physical CPU cores.
- Validate that the processor CPU sub-elements are correct.
- Validate that at least one I/O APIC device is present.
- Validate that all I/O APICs have a valid source ID capability.
- Validate that at least one and at most eight IOMMU devices are present.
- Validate that all IOMMUs have the AGAW capability set correctly and that multiple IOMMUs specify the same value.
- Validate that all IOMMUs have correct register offset capabilities.
- Check that the hardware contains a system board device providing the expected configuration.

6.7 Kernel

The following kernel-specific checks are performed on the policy.

- Validate that all CPU-local data section virtual addresses are equal.
- Validate that all CPU-local BSS section virtual addresses are equal.
- Validate that all global data section virtual addresses are equal and that the expected number of mappings exists.
- Validate that all stack virtual addresses are equal.
- Validate that all crash audit mappings exist and that their virtual addresses are equal.
- Validate that all (if any) microcode update virtual addresses are equal.
- Validate that every kernel has a stack and interrupt stack region mapped and both regions are guarded by unmapped pages below and above.
- Validate that all IOMMU memory-mapped IO regions are consecutive.
- Validate that each active CPU has a memory section.
- Validate that no virtual memory regions of the kernel overlap.
- Validate that the system board is referenced in the kernel logical devices section and that it provides a logical reset port.
- Validate that the debug console device and its resources matches the kernel diagnostics device specified in the platform section.

6.8 Memory

The following checks are performed to verify that the memory is correctly configured in the system policy.

- Validate that a VMXON region exists for every specified kernel.
- Validate size of VMXON regions.
- Validate that VMXON regions are in low-mem.
- Validate that all VMXON regions are consecutive.
- Validate that a VMCS region exists for each declared subject.
- Validate size of VMCS regions.
- Validate that physical memory region names are unique.
- Validate that physical memory referenced by logical memory exists.
- Validate that all physical memory addresses are page aligned.
- Validate that all virtual memory addresses are page aligned.
- Validate that all memory region sizes are multiples of page size.
- Validate kernel or subject entities encoded in physical memory names (e.g. linux|zp or kernel_0|vmxon).
- Validate that no physical memory regions overlap.
- Validate that an uncached crash audit region is present.
- Validate that there is either zero or exactly one MCU region present.

- Validate that a kernel data region exists for every CPU.
- Validate that a kernel BSS region exists for every CPU.
- Validate that a kernel stack region exists for every CPU.
- Validate that a kernel interrupt stack region exists for every CPU.
- Validate that a kernel PT region exists for every CPU.
- Validate that kernel PT regions are in the first 4G.
- Validate that scheduling info regions are mapped by the kernel running subjects of that scheduling partition. Also verify that the kernel mapping is at the expected virtual location.
- Validate that a subject state memory region with the expected size exists for every subject.
- Validate that a subject interrupts memory region with the expected size exists for every subject.
- Validate that memory of type kernel is only mapped by kernel or Tau0.
- Validate that memory of type system is not mapped by any entity.
- Validate that memory of type 'device' (e.g. device_rmrr) is only mapped by device domains.
- Validate that subject state memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject interrupts memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject MSR store memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject timed event memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject VMCS regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject FPU state regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that a subject FPU state memory region with the expected size exists for every subject.
- Validate that a subject timed event memory region with the expected size exists for every subject.
- Validate that a subject I/O Bitmap region with the expected size exists for every subject.
- Validate that a subject MSR Bitmap region with the expected size exists for every subject.
- Validate that a subject MSR store memory region exists for each subject that accesses MSR registers not managed by VMCS.
- Validate that a subject pagetable memory region exists for each subject.
- Validate that a scheduling info memory region exists for each scheduling partition.
- Validate that subjects map the scheduling info region of their associated scheduling partition.
- Validate that subject state, timed event and pending interrupts memory regions are only mapped writable by subjects in the same scheduling group or by siblings.
- Validate size of VT-d root table region.
- Validate size of VT-d context table region.
- Validate that a VT-d root table region exists if domains are present.
- Validate that a VT-d interrupt remapping table region exists.

6.9 Model Specific Registers (MSR)

The following checks are performed to verify Model Specific Register (MSR) specifications in the system policy.

- Validate that all MSR start addresses are smaller than end addresses.
- Validate that subject MSRs are in the allowed list:
  - IA32_SYSENTER_CS/ESP/EIP
  - IA32_DEBUGCTL
  - IA32_EFER/STAR/LSTAR/CSTAR/FMASK
  - IA32_FS_BASE/GS_BASE/KERNEL_GS_BASE
  - MSR_PLATFORM_INFO
  - IA32_THERM_STATUS
  - IA32_TEMPERATURE_TARGET
  - IA32_PACKAGE_THERM_STATUS
  - MSR_RAPL_POWER_UNIT
  - MSR_PKG_POWER_LIMIT
  - MSR_PKG_ENERGY_STATUS
  - MSR_DRAM_ENERGY_STATUS
  - MSR_PP1_ENERGY_STATUS
  - MSR_CONFIG_TDP_CONTROL
  - IA32_PM_ENABLE
  - IA32_HWP_CAPABILITIES
  - IA32_HWP_REQUEST

6.10 Platform

The following checks are performed to verify the correctness of the platform configuration in the system policy.

- Validate that physical devices referenced by device aliases exist.
- Validate that physical device resources referenced by device aliases exist.
- Validate that physical devices referenced by device classes exist.
- Validate that subject devices that reference an alias only contain resources provided by the device alias.
- Validate that the physical device and resources referenced by the kernel diagnostics device exists.
- Validate that the kernel diagnostics device resources match the requirements of the specified diagnostics type.

6.11 Scheduling

The following checks are performed to verify the correctness of the scheduling configuration in the system policy.

- Validate that scheduling partition IDs are unique.
- Validate that scheduling group IDs are unique.
- Validate that each major frame specifies the same number of CPUs.

- Validate that scheduling partitions are scheduled in at least one minor frame and that all minor frame references are on the same logical CPU.
- Validate subject references.
- Validate that subjects are scheduled on the correct logical CPU.
- Validate that subjects are part of exactly one scheduling group.
- Validate that all subjects of a scheduling group are runnable.
- Validate tick counts in major frame.
- Validate that barrier IDs do not exceed barrier count and are unique.
- Validate that barrier sizes do not exceed the number of logical CPUs.
- Validate that the barrier sizes and count of a major frame corresponds to the minor frame synchronization points.
- Validate that minor frame barrier references are valid.
- Validate partition references in minor frames.

6.12 Subjects

The following checks are performed to verify the correctness of the subject configuration in the system policy.

- Validate subject name uniqueness.
- Validate subject CPU ID.
- Validate uniqueness of global subject IDs.
- Validate per-CPU uniqueness of local subject IDs.
- Validate memory types of memory mappings (ie. allow access by subjects).
- Validate that no subject references an IOMMU device.
- Validate that all subjects are runnable, i.e. referenced in a scheduling group.
- Validate that subject scheduling group IDs match values as determined by the scheduling plan and handover events.
- Validate that logical names of subject memory regions are unique.
- Validate that logical names of subject devices are unique.
- Validate that IRQ vector numbers of PCI device references with MSI enabled are consecutive.
- Validate that logical names of subject unmask IRQ events conform to the naming scheme (`unmask_irq_$IRQNR`) and that the unmask number matches the physical IRQ.
- Validate that no virtual memory regions of a subject overlap.
- Validate that multiple initramfs regions are consecutive.
- Validate that no subject has write access to the crash audit region.
- Validate that subject device mmconf mappings are correct.
- Validate that the VMX controls conform to the checks specified in Intel SDM Vol. 3C, "26.2.1 Checks on VMX Controls".
- Validate that the Pin-Based VM-Execution controls meet the requirements for the execution of Muen.

- Validate that the Processor-Based VM-Execution Controls meet the requirements for the execution of Muen.
- Validate that the secondary Processor-Based VM-Execution Controls meet the requirements for the execution of Muen.
- Validate that the VM-Exit Controls meet the requirements for the execution of Muen.
- Validate that the VM-Entry Controls meet the requirements for the execution of Muen.
- Validate that the VMX CR0 guest/host masks meet the requirements for the execution of Muen.
- Validate that the VMX CR4 guest/host masks meet the requirements for the execution of Muen.
- Validate that the VMX Exception bitmap meet the requirements for the execution of Muen.

Chapter 7

Policy Structure

7.1 Policy Schema Documentation

7.1.1 systemType

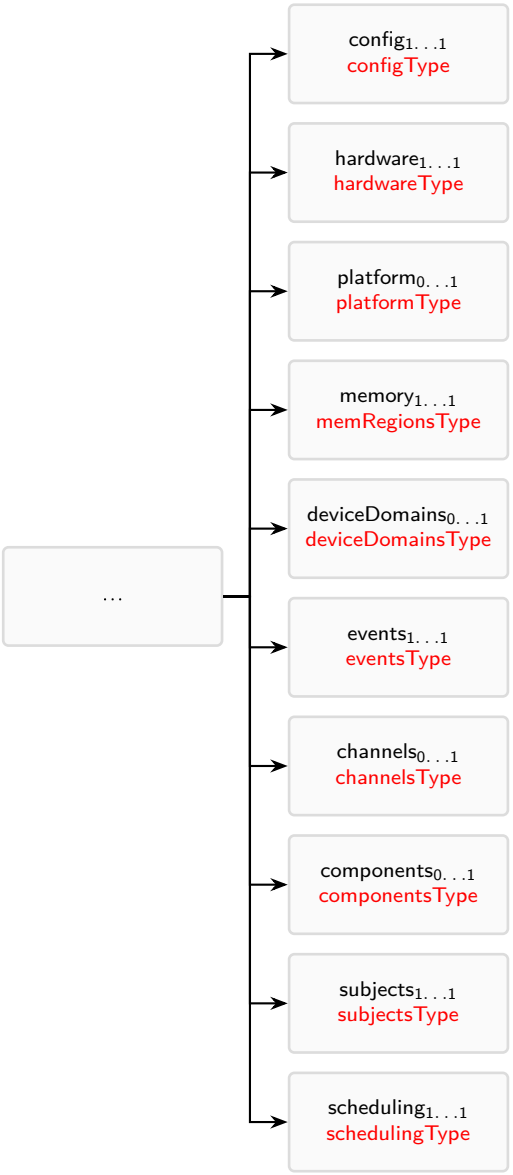
A Muen system policy specifies all hardware resources such as physical memory, devices, CPU time, etc and how these resources are accessed by the separation kernel, the subjects and devices.

The `system` section is the top-level element in the Muen system policy. It contains various sub-elements which specify all aspects of a concrete system.

This is the *source format* of the Muen system policy. It allows for abstractions, such as channels, which are broken down into their constituent parts by the toolchain in format A and B accordingly.

See line 3 and following in listing 8.1 on page 105 for an annotated system policy example.

Structure



7.1.2 configType

The purpose of a config section is to specify configuration values which parameterize a system or a component. It allows to declare boolean, string and integer values. The following sections in the system policy provide support for configuration values:

- System
- Platform
- Component

During the build process, configuration values provided by the platform are merged into the global system configuration. Component configuration values allow the parameterization of component-local functionality.

Besides component parameterization, configuration options can be used in `if` conditionals, as shown in the following example.

```
1    <if variable="xhcidbg_enabled" value="true">
      ...
3    </if>
```

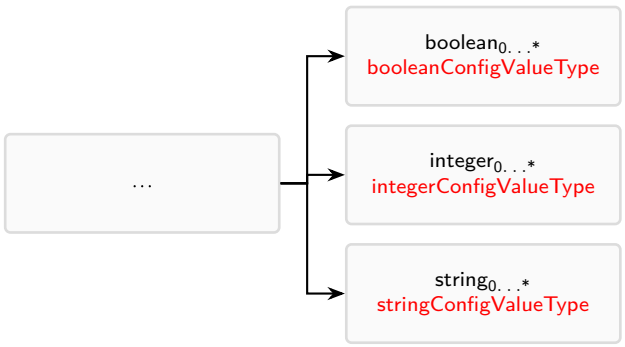
If the type of the referenced variable is 'string' the comparison is case-sensitive. A second use case is XML attribute value expansion as follows:

```
1    <channel name="debuglog" size="$logchannel_size"/>
```

The `size` attribute value is not specified directly, but parameterized via an integer configuration option.

See line 17 in listing 8.1 for an example config section.

Structure



7.1.3 booleanConfigValueType

Configuration option for values in boolean format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	booleanType	optional
Value of the configuration option.		

7.1.4 nameType

Base: xs:string

The `nameType` is used to give (unique) names to elements.

Restrictions

minimal length = 1, maximal length = 63

7.1.5 booleanType

Base: xs:string

Boolean type.

Restrictions

values:

- true
- false

7.1.6 integerConfigValueType

Configuration option for values in integer format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	xs:integer	optional
Value of the configuration option.		

7.1.7 stringConfigValueType

Configuration option for values in string format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	xs:string	optional
Value of the configuration option.		

7.1.8 hardwareType

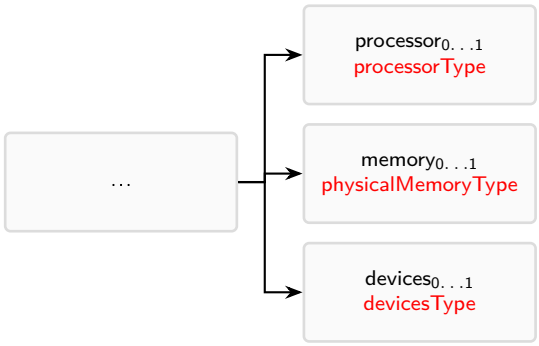
Systems running the Muen SK perform static resource allocation at integration time. This means that all available hardware resources of a target machine must be defined in the system policy in order for these resources to be allocated to subjects.

The `hardware` element is the top-level element of the hardware specification in the system policy. Information provided by a hardware description includes the amount of available physical memory blocks including reserved memory regions (RMRR), the number of physical CPU cores and hardware device resources.

The Muen toolchain provides a handy tool to automate the cumbersome process of gathering hardware resource data from a running Linux system: `mugenhwcfg`[1].

See line 81 in listing 8.1 for an example hardware section.

Structure



7.1.9 processorType

The `processor` element specifies the number of CPU cores, the processor speed in kHz and the Intel VMX preemption timer rate.

Since Intel CPUs can have arbitrary APIC identifiers, the APIC IDs of all physical CPUs are enumerated here. The APIC ID is required for interrupt and IPI routing.

[1] https://git.codelabs.ch/?p=muen/mugenhwcfg.git

The `processor` element also lists register values for all CPUID leaves of the hardware target, and some MSR values of interest.

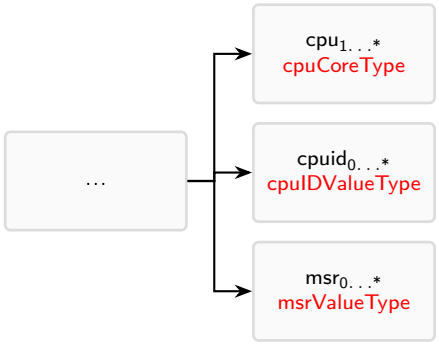
See line 100 in listing 8.1 for an example processor element. The `cpu` elements must fulfill the following constraints to be valid:

- A node exists for every physical core of the system
- The optional `cpuId` attribute of all elements must be consecutive
- If specified, a node with `cpuId` value `0` must exist
- A node with `apicId` value `0` must exist and, if specified, it must have a `cpuId` value within the active CPU range, i.e. the BSP is part of the system scheduling plan
- All `apicId` attributes must have even numbers

Attributes

Name	Type	Use
cpuCores	xs:positiveInteger	required
	Number of available CPU cores. Note that this value designates physical, hardware cores, not Hyper-Threading (HT) *threads*. HT is disabled on Muen.	
speed	xs:positiveInteger	required
	Tick rate of CPU cores in kHz.	
vmxTimerRate	vmxTimerRateType	required
	The VMX-premption timer counts down at a rate proportional to that of the timestamp counter (TSC). This value specifies this proportion, see Intel SDM Vol. 3C, "25.5.1 VMX-Preemption Timer" for more details.	

Structure



7.1.10 cpuCoreType

Specification of one physical CPU core.

Attributes

Name	Type	Use
apicId	xs:unsignedByte	required
	CPU local APIC ID, see Intel SDM Vol. 3A, "10.4.6 Local APIC ID".	
cpuId	xs:unsignedByte	optional
	Unique CPU ID.	

7.1.11 cpuIDValueType

Register values for a CPUID leaf, see Intel SDM Vol. 2A, "3.2 Instructions (A-L)", CPUID.

Attributes

Name	Type	Use
leaf CPUID leaf.	word32Type	required
subleaf CPUID subleaf.	byteType	required
eax EAX register value of this leaf.	word32Type	required
ebx EBX register value of this leaf.	word32Type	required
ecx ECX register value of this leaf.	word32Type	required
edx EDX register value of this leaf.	word32Type	required

7.1.12 word32Type

Base: word64Type < xs:string
32-bit machine word.

Restrictions

value $\leq$ 13

7.1.13 byteType

Base: xs:string
Machine octet (8-bits).

Restrictions

Pattern = 16#[0-9a-fA-F]2#

7.1.14 msrValueType

Register value of an MSR of interest.

Attributes

Name	Type	Use
address MSR address.	word32Type	required
name Name of MSR.	xs:string	required
regval Register value.	word64Type	required

7.1.15 word64Type

Base: xs:string
64-bit machine word.

Restrictions

Pattern = 16#[0-9a-fA-F]4(_([0-9a-fA-F]4))0,3#

7.1.16 vmxTimerRateType

Base: xs:nonNegativeInteger
VMX-preemption timer count down rate.

Restrictions

value ≤31

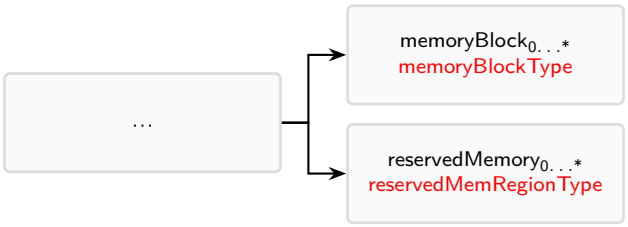
7.1.17 physicalMemoryType

The hardware `memory` element specifies the available physical memory blocks including reserved memory regions (RMRR, see Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure").

Only memory blocks reported by the BIOS E820 map as non-*reserved* must be configured in this section, e.g. *usable* or *ACPI NVS*, *ACPI data*.

See line 165 in listing 8.1 for an example memory element.

Structure



7.1.18 memoryBlockType

Base: memoryBlockBaseType
Consecutive block of memory provided by the hardware.

Attributes

Name	Type	Use
name	nameType	required
	Name of memory block.	
physicalAddress	word64Type	required
	Start address of memory block.	
size	memorySizeType	required
	Size of memory block.	
allocatable	booleanType	optional
	Indication to a physical memory allocator that this block allows allocation of physical memory regions. If this attribute is false, an allocator should only place fixed memory regions in this range, i.e. memory regions with the physicalAddress attribute set by the integrator. Note that host physical memory below 1 MiB is considered special, the attribute must be set to false. Only unmapped memory of type *system* is allowed in that special memory block.	

7.1.19 memorySizeType

Base: word64Type < xs:string

The `memorySizeType` is used to declare memory sizes.

Restrictions

no restriction

7.1.20 reservedMemRegionType

Base: memoryBlockBaseType

A `reservedMemory` element is a special memory block declaration. It specifies a reserved memory region as outlined in the Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure" (RMRR).

Reserved memory regions are BIOS allocated memory ranges that may be DMA targets for certain legacy device use-cases. Devices that require access to such a region refer to it by name.

See line 181 in listing 8.1 for an example RMRR element.

Attributes

Name	Type	Use
name	nameType	required
Name of memory block.		
physicalAddress	word64Type	required
Start address of memory block.		
size	memorySizeType	required
Size of memory block.		

7.1.21 devicesType

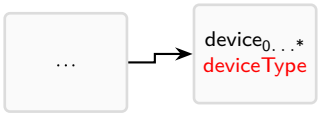
The `devices` element enumerates all devices provided by the hardware platform. Different kinds of devices, be it PCI(e) or legacy (non-PCI), can be declared in this section.

See line 194 in listing 8.1 for an example devices enumeration.

Attributes

Name	Type	Use
pciConfigAddress	word64Type	optional
Physical base address of the PCI configuration space region.		
pciConfigSize	word64Type	optional
Size of the PCI configuration space region.		

Structure



7.1.22 deviceType

Base: deviceBaseType

The `device` element specifies a physical device and its associated resources. There are three main device resource types:

- IRQ
- I/O port range
- Memory

The presence of a PCI element indicates whether the device is a PCI or a legacy device.

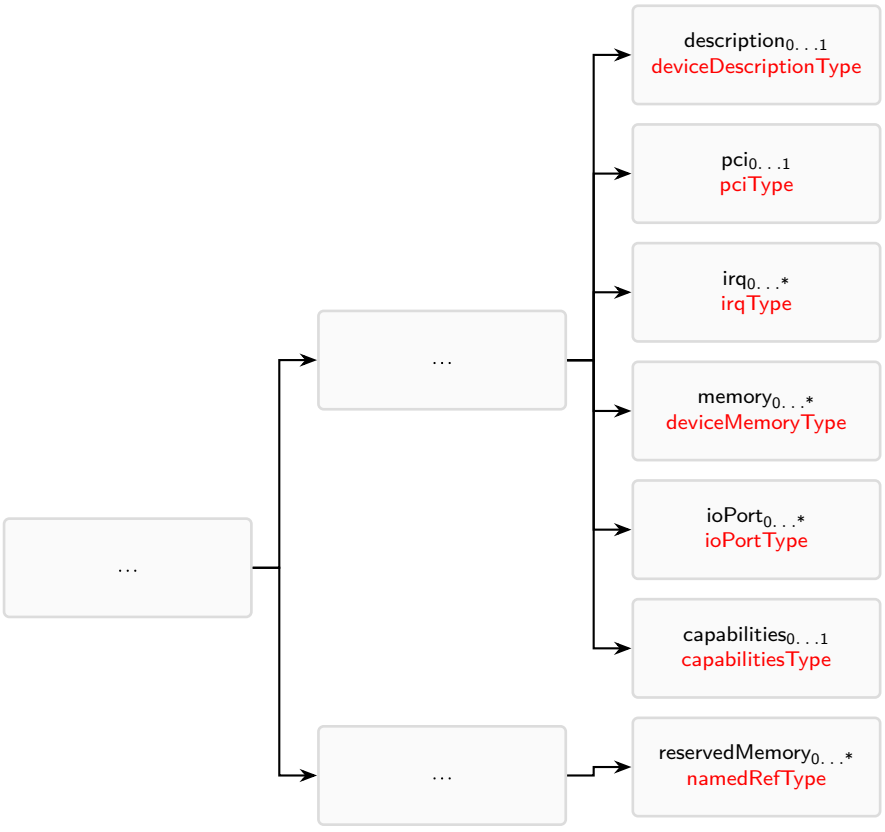
Capabilities can be used to convey additional device-specific information. The base address of the memory mapped PCI config space is defined by the `pciConfigAddress` attribute.

See line 200 in listing 8.1 for an example device declaration.

Attributes

Name	Type	Use
name Unique device name.	nameType	required

Structure



7.1.23 namedRefType

The `namedRefType` is used to reference a named element in the policy.

Attributes

Name	Type	Use
ref Name of referenced element.	nameType	required

7.1.24 deviceDescriptionType

Base: xs:string

Device description (free text).

Restrictions

no restriction

7.1.25 pciType

Base: pciAddressType

PCI(e) devices are specified using the `pci` element.
The element provides the following information:

- PCI device address (BDF)
- Identification
- IOMMU group information

The location of the PCI device in the PCI topology is specified by the Bus, Device, Function triplet (BDF).
See line 334 in listing 8.1 for an example PCI element declaration.

Attributes

Name	Type	Use
bus PCI Bus number.	byteType	required
device PCI Device number.	pciDeviceNumberType	required
function PCI Function number.	pciFunctionNumberType	required

Structure

7.1.26 deviceIdentificationType

The `identification` element specifies the PCI device class, device, revision and vendor ID.

For more information, consult the PCI Local Bus Specification, "Configuration Space Decoding".

See line 349 in listing 8.1 for an example PCI identification.

Attributes

Name	Type	Use
classcode PCI device class.	word16Type	required
vendorId PCI vendor ID.	word16Type	required
deviceId PCI device ID.	word16Type	required
revisionId PCI device revision ID.	byteType	required

7.1.27 word16Type

Base: word64Type < xs:string

16-bit machine word.

Restrictions

length = 8

7.1.28 iommuGroupType

Devices in the same IOMMU group cannot be properly isolated from each other because they may perform inter-device transactions directly, without going through the IOMMU.

Note that this information is currently not used by the toolchain. It is a hint to the system integrator whether two devices can be properly isolated from each other or not.

See line 358 in listing 8.1 for an example IOMMU group declaration.

Attributes

Name	Type	Use
id IOMMU group number.	xs:nonNegativeInteger	required

7.1.29 pciDeviceNumberType

Base: xs:string
PCI Device number.

Restrictions

Pattern = 16#[0|1][0-9a-fA-F]#

7.1.30 pciFunctionNumberType

Base: xs:nonNegativeInteger
PCI Function number.

Restrictions

value ≤7

7.1.31 irqType

The `irq` element specifies a device IRQ resource.
The specified IRQ number is one of:

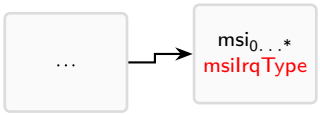
- Legacy IRQ (ISA)
  Range `0 .. 15`.
- PCI INTx IRQ, line-signaled
  Range `0 .. Max_LSI_IRQ`, whereas `Max_LSI_IRQ` is defined by the hardware I/O APIC configuration `gsi_base` + `max_redirection_entry` of I/O APIC with `max(gsi_base)`. `gsi_base` and `max_redirection_entry` are I/O APIC device capabilities.

`msi` sub-elements are present if the device supports MSI interrupts. The element count designates the number of supported MSI interrupts.
See line 237 in listing 8.1 for an example device IRQ declaration.

Attributes

Name	Type	Use
name Name of device IRQ resource.	nameType	required
number Legacy or PCI line-based IRQ.	irqNumberType	required

Structure



7.1.32 msiIrqType

There are two different interrupt types which devices may trigger: legacy/PCI LSI IRQs and Message Signaled Interrupts (MSI). The legacy/PCI LSI IRQ is specified by the number attribute of the `irq` element. For MSIs, each `msi` element defines an MSI IRQ that may be assigned to subjects. Each MSI may be individually routed.

See line 400 in listing 8.1 for example device MSI elements.

Attributes

Name	Type	Use
name	nameType	required
Name of MSI resource.		

7.1.33 irqNumberType

Base: xs:nonNegativeInteger

IRQ number. High IRQs are reserved for kernel usage.

Restrictions

value ≤220

7.1.34 deviceMemoryType

Base: memoryBlockBaseType

A device `memory` element specifies a memory region which is used to interact with the associated device.

For PCI devices, the specified region is programmed into one device BAR (Base Address Register) by system firmware. See the PCI Local Bus Specification or the PCI Express Base Specification for more details.

See line 218 in listing 8.1 for an example device memory declaration.

Attributes

Name	Type	Use
name	nameType	required
Name of memory block.		
physicalAddress	word64Type	required
Start address of memory block.		
size	memorySizeType	required
Size of memory block.		
caching	cachingType	required
Device memory caching type.		

7.1.35 cachingType

Base: xs:string

Memory caching type, see Intel SDM Vol. 3A, "11.3 Methods of Caching Available".

- Strong Uncacheable (UC)
- Write Combining (WC)
- Write Through (WT)
- Write Back (WB)
- Write Protected (WP)

Restrictions

Pattern = UC|WC|WT|WB|WP

7.1.36 ioPortType

The `ioPort` element specifies a device I/O port resource from `start` octet up to and including `end` octet. A single byte-accessed port is designated by specifying the same `start` and `end` values.

See line 228 in listing 8.1 for an example device IRQ declaration.

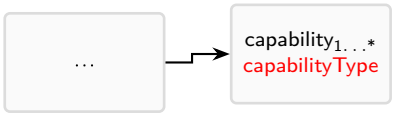
Attributes

Name	Type	Use
name Name of I/O port resource.	nameType	required
start Start port of this resource.	word16Type	required
end End port of this resource.	word16Type	required

7.1.37 capabilitiesType

List of device capabilities.

Structure



7.1.38 capabilityType

Base: xs:string

A device `capability` is used to assign additional information to a device. Such a capability might be used by the Muen toolchain to perform certain actions on devices with a given capability (e.g. `ioapic`). A system integrator may use this facility to define its own capabilities used by custom tools.

A capability element can have an optional value.

See line 290 in listing 8.1 for example capabilities.

Attributes

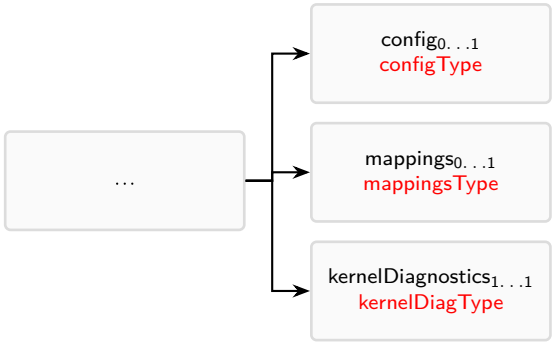
```
Name            Type        Use
-------------------------------------------------------------------------------
name            xs:string   required
  Capability name (free text).
```

7.1.39 platformType

To enable a uniform view of the hardware resources across different physical machines from the system integrators perspective, the platform description layer is interposed between the hardware resource description and the rest of the system policy. This allows to build a Muen system for different physical target machines using the same system policy.

See line 556 in listing 8.1 for an example platform section.

Structure

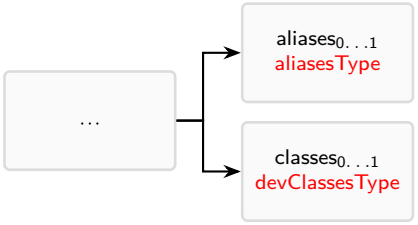


7.1.40 mappingsType

Platform device alias and class mappings section. Used to assign a stable name to a hardware device or to group (multiple) devices under a given name.

See line 565 in listing 8.1 for an example platform mappings section.

Structure



7.1.41 aliasesType

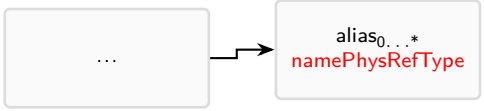
Aliases are a renaming mechanism for physical hardware devices and their resources. By using alias names in the system policy references to concrete hardware resources can be avoided. Additionally, aliases may be used to define a device which only contains a subset of the resources of the physical device. This can be achieved by only renaming the resources that the device alias should export.

See line 571 in listing 8.1 for an example aliases section.

Aliases are resolved in the following system policy sections.

- `/system/subjects/subject/component/map`
- `/system/subjects/subject/devices/device`
- `/system/deviceDomains/domain/devices/device`

Structure



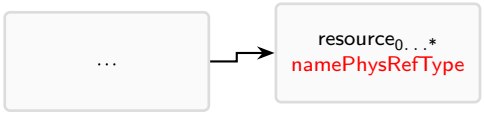
7.1.42 namePhysRefType

Named resource reference. Used for device aliases and device alias resource references.

Attributes

Name	Type	Use
name	nameType	required
Alias name.		
physical	nameType	required
Reference to physical device or device resource.		

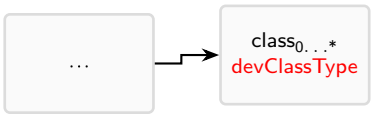
Structure



7.1.43 devClassesType

The `classes` element specifies a list of device classes.

Structure



7.1.44 devClassType

Device classes enable the grouping of devices and allow referencing all devices by a single name. This simplifies the process of assigning multiple devices to a subject.

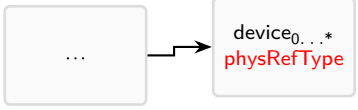
Note: A device class may contain an arbitrary number of devices, including zero.

See line 603 in listing 8.1 for a device class example.

Attributes

Name	Type	Use
name	nameType	required
Device class name.		

Structure



7.1.45 physRefType

Reference to physical device or physical device resource.

Attributes

Name	Type	Use
physical	nameType	required
Physical resource name (device or resource sub-element).		

7.1.46 kernelDiagType

The debug build Muen SK can be instructed to output debugging information during runtime. The platform diagnostics device specifies which device the kernel is to use for this purpose.

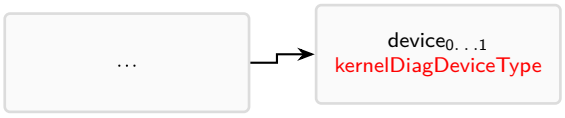
The presence of this device and the necessary resources are checked by the validator tool.

See line 623 in listing 8.1 for an example platform diagnostic device configuration.

Attributes

Name	Type	Use
type	kernelDiagKindType	required
Specifies the type of diagnostics device to use.		

Structure



7.1.47 kernelDiagDeviceType

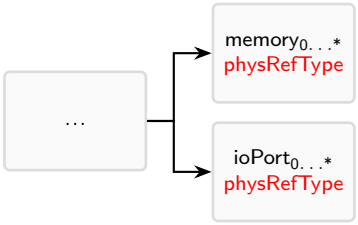
Reference to physical device for `uart` and `vga` diagnostic device type.

If an UART device is referenced via type `uart`, an I/O port resource must be provided. If a VGA device is referenced via type `vga`, a memory resource must be provided (both checked via validator).

Attributes

Name	Type	Use
physical	nameType	required
Name of physical device to use for kernel diagnostics output.		

Structure



7.1.48 kernelDiagKindType

Base: xs:string

Type of diagnostics device. While `none` disables kernel diagnostics output, `uart` specifies an Universal Asynchronous Receiver-Transmitter serial device. `hsuart` is a High-Speed UART with memory mapped I/O.

`vga` outputs the kernel diagnostics information to a VGA console, which is mainly useful for initial bring-up of a new hardware platform with no UART device.

Restrictions

values:

- none
- uart
- hsuart
- vga

7.1.49 memRegionsType

This section declares all physical memory regions (RAM) and thus the physical memory layout of the system. Regions declared in this section can be assigned to subjects and device domains.

Memory regions are defined by the following attributes:

- Name
- Caching type
- Size
- Physical address*
- Alignment*
- Memory type*

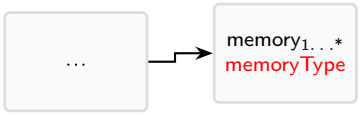
Attributes with an asterisk are optional. While alignment and memory type are set to a default value if not specified, the physical address is filled in by the allocator tool, which allocates all memory regions and finalizes the physical memory layout.

Additionally, the content of a region can be declared as backed by a file or filled with a pattern.

Note: The caching type is an attribute of the physical memory region by design to avoid inconsistent typing, even though the Intel Page Attribute Table (PAT) mechanism allows to set it for each memory mapping, see Intel SDM Vol. 3A, "11.12.4 Programming the PAT".

See line 637 in listing 8.1 for an example memory region section.

Structure



7.1.50 memoryType

Base: physicalMemBaseType < memoryBaseType

The `memoryType` specifies a physical memory region by name, size and caching.

If no explicit physical address is specified for the region, the `mucfgalloc` tool will allocate a free one in usable memory, honoring the optional `alignment` attribute.

If no explicit `alignment` attribute is specified, it is set to `16#1000#` by the expander. If no explicit `type` attribute is specified, it is set to `subject` by the expander.

Attributes

Name	Type	Use
size	memorySizeType	required
Size of region. Must be a multiple of page size (4K). Enforced by validator.		
name	nameType	required
Name of region.		
caching	cachingType	required
Caching type to use for memory region.		

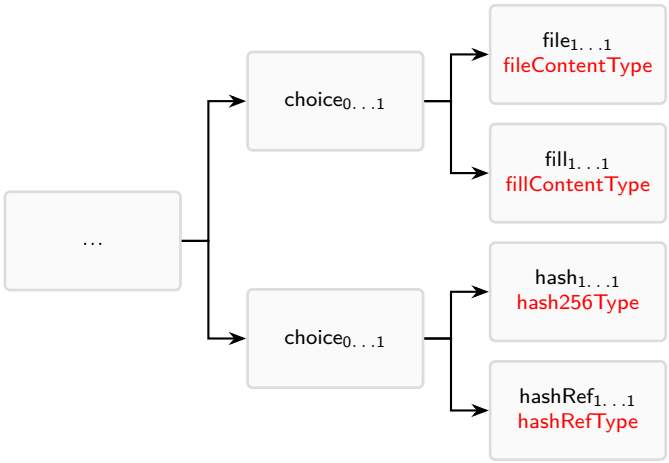
```
(continuation)
Name              Type                Use

type              srcMemoryKindType   optional
 Optional subject memory type.

alignment         alignmentType       optional
 Alignment the physical address of the memory region must honor (checked by the
 validator tool).

physicalAddress   word64Type          optional
 Physical address of memory region.
```

Structure



7.1.51 srcMemoryKindType

Base: memoryKindType < xs:string

Memory types allowed in policy format source physical memory section. For information about subject memory types, see 7.1.120.

Besides subject types, the following memory types are allowed:

- `kernel_microcode`
  Memory region designating a CPU microcode update, e.g. added by the `mucfgucode` tool (5.5.5).

Restrictions

values:

- kernel_microcode
- subject
- subject_info
- subject_state
- subject_binary
- subject_channel
- subject_crash_audit
- subject_initrd
- subject_bios
- subject_acpi_rsdp
- subject_acpi_xsdt

- subject_acpi_fadt
- subject_acpi_dsdt
- subject_zeropage
- subject_solo5_boot_info
- subject_device
- subject_timed_event

7.1.52 alignmentType

Base: word64Type < xs:string

Memory alignment constraint for memory region. Taken into account by the allocator tool and checked by the validator.

Restrictions

values:

- 16#1000#
- 16#0020_0000#
- 16#4000_0000#

7.1.53 fileContentType

The `file` child element designates a file-backed memory region.

The `filename` attribute specifies the name of the file to use as content for the physical memory region, the `offset` attribute is `none` by default but can be customized to include a partial file.

See line 714 in listing 8.1 for a file-backed memory region example. The following checks on the file content are performed.

- If `offset` is `none`, the size of the file must be less than the memory region size.
- If `offset` is not `none`, the offset must be less than the file size. The file size is *not* checked but the memory region size is used as upper bound.

Attributes

Name	Type	Use
filename	xs:string	required
	Filename of file to (partially) include. Note that the actual file processed by the toolchain also depends on the working directory passed as command line option to the specific tool.	
offset	optionalOffsetType	required
	Read file offset in bytes.	

7.1.54 optionalOffsetType

Optional file offset value in bytes.

Restrictions

Union of

- word64Type
- noneType

7.1.55 fillContentType

The `fill` element designates a memory region which is initialized with the given pattern.
See line 671 in listing 8.1 for a file-backed memory region example.

Attributes

Name	Type	Use
pattern	byteType	required
Fill pattern (hex).		

7.1.56 hash256Type

The `hash` child element of a memory region designates a 256-bit hash over the memory content.
The Mucfgmemhashes tool in the Muen toolchain generates such a hash-sum for every content-backed memory region in a given policy.

Attributes

Name	Type	Use
value	optionalHashType	required
256-Bits message digest over file-backed memory content.		

7.1.57 optionalHashType

Allows the specification of a hash digest or `none`.

Restrictions

Union of

- hash256DigestType
- noneType

7.1.58 hashRefType

The optional `hashRef` child element of a physical memory region instructs the Mucfgmemhashes tool to copy the hash element of the referenced memory region after message digest generation.
From an abstract point of view, the `hashRef` element is a way to link multiple memory regions by declaring that the hash of the content is the same. This concept is e.g. used by the subject loader mechanism to restore writable memory regions to their initial state.

Attributes

Name	Type	Use
memory	nameType	required
Name of referenced physical memory region.		

7.1.59 deviceDomainsType

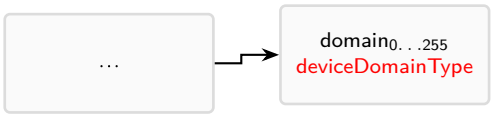
The physical memory accessible by PCI devices is specified by so called device domains. Such domains define memory mappings of physical memory regions for one or multiple devices. Device references select a subset of hardware devices provided by the hardware/platform. Devices may be referenced by device name, alias or device class.

Device references can optionally set the `mapReservedMemory` attribute so RMRR regions referenced by the device are also mapped into the device domain.

Device domains are isolated from each other by the use of Intel VT-d.

See line 734 in listing 8.1 for a device domain example.

Structure



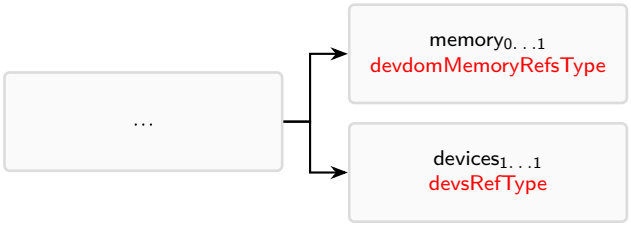
7.1.60 deviceDomainType

A device domain allows referenced devices access to the specified memory regions. It also provides handling for reserved memory region reporting (RMRR), see Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure".

Attributes

Name	Type	Use
name	nameType	required
Name of the device domain.		

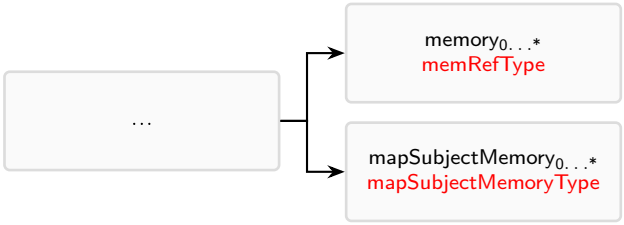
Structure



7.1.61 devdomMemoryRefsType

List of physical memory region references and optional map subject memory elements.

Structure



7.1.62 memRefType

A `memory` element maps a physical memory region into the address space of a device domain or subject entity. The region will be accessible to the entity at the specified `virtualAdress` with permissions defined by the `executable` and `writable` attributes.

See line 759 in listing 8.1 for an example of such a mapping.

Attributes

Name	Type	Use
virtualAddress	word64Type	required
Address in entity address space where the physical memory region is mapped.		
physical	nameType	required
Name of referenced physical memory region.		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

7.1.63 mapSubjectMemoryType

This element instructs the expander to map memory regions of the specified subject into the device domain. Only regions which are writable and of type *Subject* and *Subject_Initrd* are mapped.

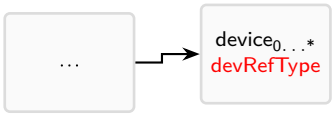
Attributes

Name	Type	Use
subject	nameType	required
Name of the subject.		
virtualAddressOffset	word64Type	optional
Optional offset value. If this attribute it specified, the given value will be added to the *virtualAddress* value of a mapped memory region.		

7.1.64 devsRefType

Device domain device references.

Structure



7.1.65 devRefType

Device domain device reference. Referenced devices gain access to memory regions of device domain.

Attributes

Name	Type	Use
logical	nameType	required
Logical name in this context.		
physical	nameType	required
Physical device or device alias to include in given device domain.		

```
(continuation)
Name                Type          Use

mapReservedMemory   booleanType   optional
  Whether to automatically map RMRR memory associated with device.
```

7.1.66 eventsType

Events are an activity caused by a subject (source) that impacts a second subject (target) or is directed at the kernel. Events are declared globally and have a unique name to be unambiguous. An event must have a single source and one target.

Subjects can use events to either deliver an interrupt, hand over execution to or reset the state of a target subject. The first kind of event provides a basic notification mechanism and enables the implementation of event-driven services. The second type facilitates suspension of execution of the source subject and switching to the target. Such a construct is used to pass the thread of execution on to a different subject, e.g. invocation of a debugger subject if an error occurs in the source subject. The third kind is used to facilitate the restart of subjects.

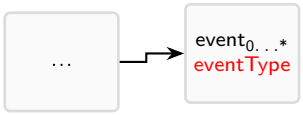
An event can also have the same source and target, which is called *self* event. Such events are useful to implement para-virtualized timers in VM subjects for example.

Kernel events are special in that they are targeted at the kernel. The currently supported events are system reboot and shutdown.

For documentation about linking physical events to source and target subjects, see section 7.1.133.

See line 775 in listing 8.1 for an example events section.

Structure



7.1.67 eventType

The `eventType` specifies an event by name and mode.

The following event modes are currently supported:

- `asap`
  The asap event is an abstraction to state that the event should be delivered as soon as possible, depending on the CPU of the target subject. If the target runs on another CPU core, this mode is expanded to mode *ipi*, which is only available in policy formats A and B, instructing the kernel to preempt the kernel running the target subject and inject the event immediately. If the target subject runs on the same core as the source subject, the mode is expanded to mode *async*.

- `async`
  Async events trigger no preemption at the target subject. The event is marked as pending in the target subject's pending event table and inserted on the next VM exit/entry cycle of the target subject.

- `self`
  An event can also have the same source and target, which is called a self event. Such events are useful to implement para-virtualized timers in VM subjects for example. A subject sends itself a delayed event, using the timed event mechanism. Note that a self event must always have a target action assigned, which is checked by the validator.

- `switch`
  The switch mode facilitates suspension of execution of the source subject and switching to the target. This can only happen between subjects running on the same core. Such a construct is used to pass the thread of execution on to a different subject, e.g. invocation of a debugger subject if an error occurs in the source subject. It is called *handover* or *handover event*.

- `kernel`
  These kinds of events are directed at the kernel an thus only specify a source since the target is the kernel. They are used to enable specific subjects to unmask level-triggered IRQs and trigger a system reboot, poweroff or explicit panic (crash audit slot allocation and reboot).

See line 799 in listing 8.1 for an example global event declaration.

Attributes

Name	Type	Use
name	nameType	required
Name of the event.		
mode	eventModeType	required
Mode of the event.		

7.1.68 eventModeType

Base: xs:string

Event mode.

See 7.1.67 for details about the supported event modes.

Restrictions

values:

- asap
- async
- self
- switch
- kernel

7.1.69 channelsType

Inter-subject communication is specified by so called channels. These channels represent directed information flows since they have a single writer and possibly multiple readers. Optionally a channel can have an associated notification event (doorbell interrupt).

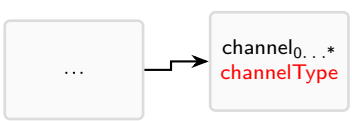
Channels are declared globally and have an unique name to be unambiguous.

Note that channels are a policy source format abstraction. The toolchain resolves this concept into memory regions and events as well as the appropriate subject mappings.

For documentation about linking physical channels to subjects see section 7.1.141. For documentation about declaring requested channels in components see section 7.1.102. For information how to map a physical channel with a logical component channel at subject level, see section 7.1.150.

See line 865 in listing 8.1 for an example channel section.

Structure



7.1.70 channelType

The `channel` element declares a physical channel.

Besides the `name` and `size` of the channel, the optional `hasEvent` attribute can be set to declare that the given channel requests an associated event. The expander tool will then automatically create a global event of the requested event type.

See line 879 in listing 8.1 for an example channel declaration.

Attributes

Name	Type	Use
name Channel name.	nameType	required
size Size of the channel in bytes. Must be a multiple of page size (4K). Enforced by validator.	memorySizeType	required
hasEvent Associated event type (if any).	eventModeType	optional

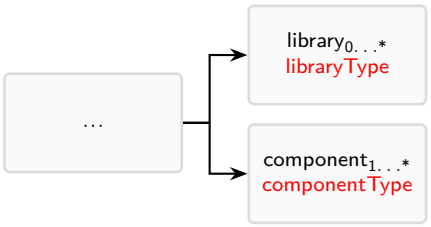
7.1.71 componentsType

The `components` element holds a list of components and component libraries.

Note that components are a policy source format abstraction. The toolchain resolves this concept into subjects by adding the appropriate memory regions, events and devices.

See line 914 in listing 8.1 for an example components section.

Structure



7.1.72 libraryType

A component library is a specialized component specification which is used to share common resources required for library code to operate. Component libraries can be included by multiple components in order to share functionality. An example is a logging service provided by a dedicated component, whereas the logging client is provided as a library with a shared memory channel for the actual log messages.

A component specification declares library dependencies to request the library resources from the system through the inclusion of the library specification in the `depends` section. This way components inherit the resources of libraries.

On the source code level, a library is included by mechanisms provided by the respective programming language. Note that the component library code is *not* shared between components but lives in the isolated execution environment of a subject instantiating the component (i.e. statically linked libraries).

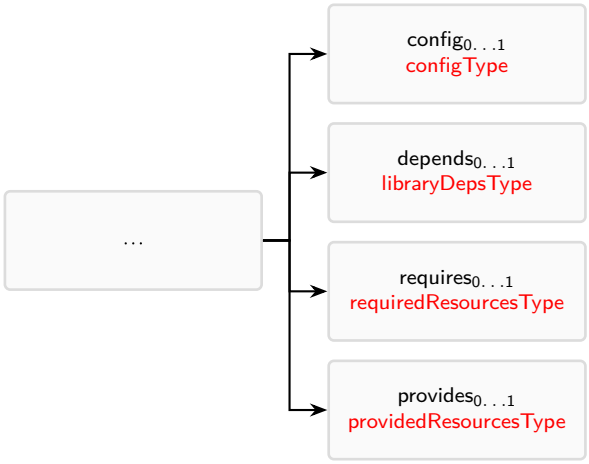
Libraries can request the same resources as ordinary components. A subject instantiating the component must also map the resources requested by libraries the component depends on.

See line 923 in listing 8.1 for example library specifications.

Attributes

Name	Type	Use
name Component/library name.	nameType	required

Structure



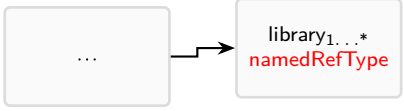
7.1.73 libraryDepsType

Components and libraries are allowed to declare dependencies to other libraries. All resources required by the included library are merged with the ones specified by the component or library. Libraries can depend on other libraries.

A subject realizing this component must correctly map all component and library resource requirements to physical resources in order to fulfill the expectations.

See line 1067 in listing 8.1 for an example dependency section.

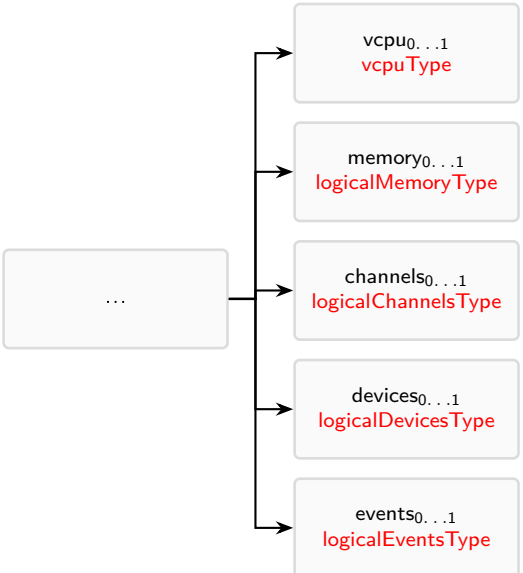
Structure



7.1.74 requiredResourcesType

Declaration of resources a component or library requires to operate.

Structure

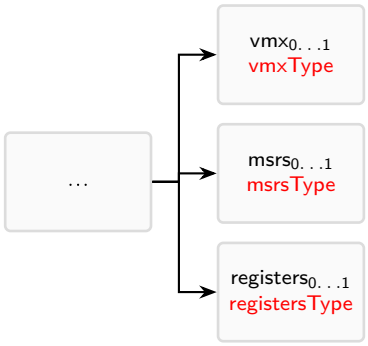


7.1.75 vcpuType

The `vcpu` element controls the execution behavior of the virtual CPU (vCPU). A default vCPU profile is selected by the component profile, but CPU execution settings can be customized both at component and subject level.

See line 1024 in listing 8.1 for an example on how to customize a vCPU profile.

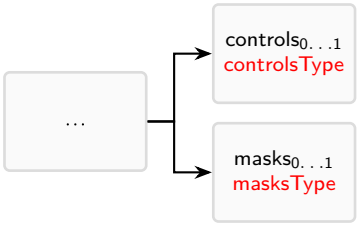
Structure



7.1.76 vmxType

Controls Intel VMX vCPU settings.

Structure

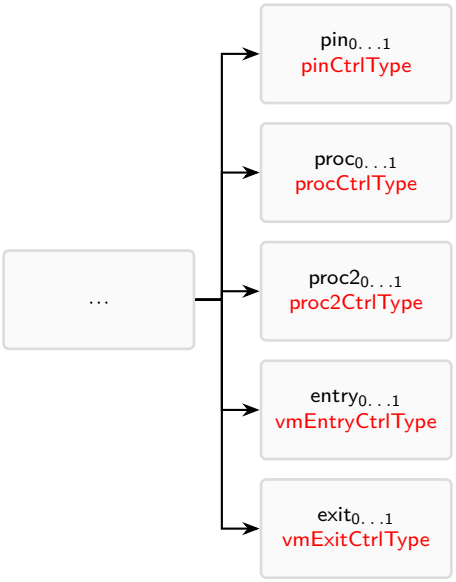


7.1.77 controlsType

Configures the following Intel VMX settings:

- Pin-Based VM-Execution Controls
- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls
- VM-Entry Controls
- VM-Exit Controls

Structure



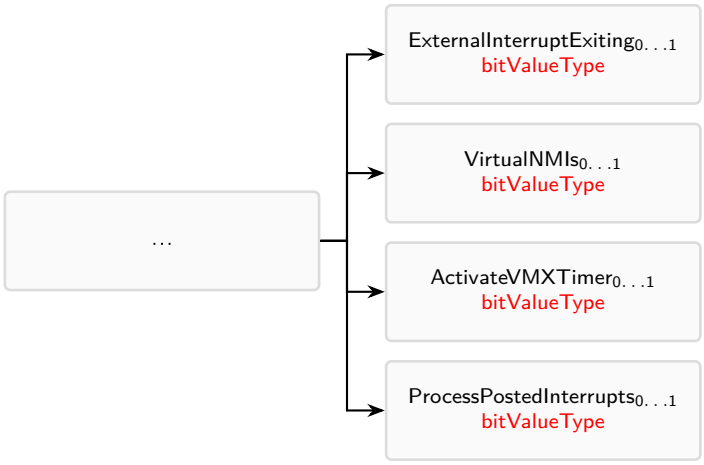
7.1.78 pinCtrlType

Configures Intel VMX pin-based VM-execution controls. These controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts) while running in VMX non-root mode.

See Intel SDM Vol. 3C, "24.6.1 Pin-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.

> ☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure



7.1.79 bitValueType

Base: xs:nonNegativeInteger

The value of one bit, either 1 (True) 0 (False).

Restrictions

$0 \leq 1$

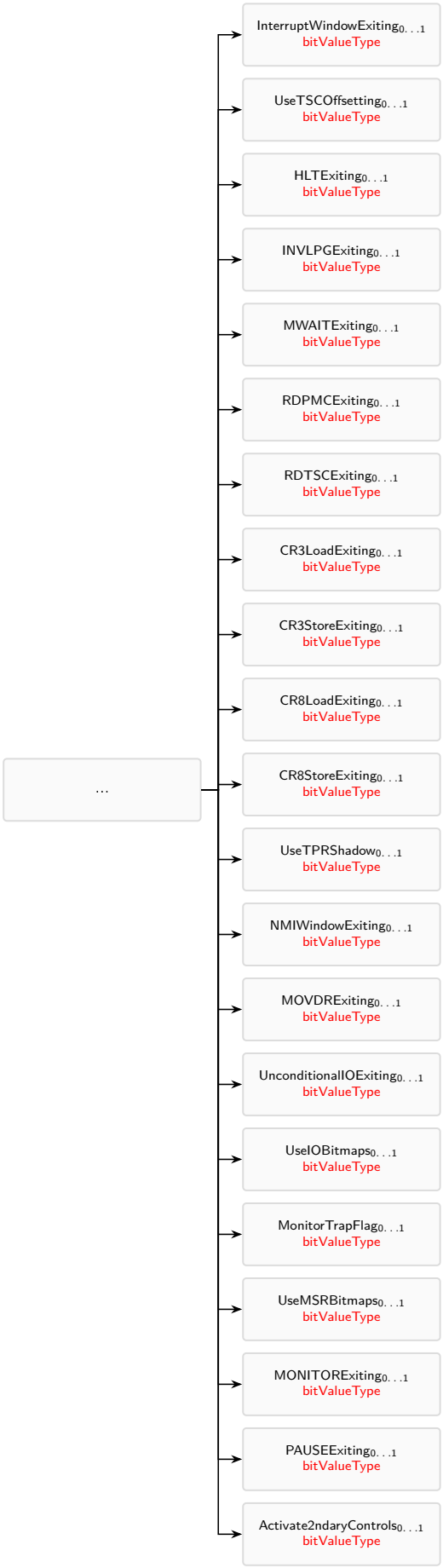
7.1.80 procCtrlType

The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions. These are the *primary processor-based* VM-execution controls and the *secondary processor-based* VM-execution controls.

The `proc` element configures the primary processor-based VM-execution controls, see Intel SDM Vol. 3C, "24.6.2 Processor-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.

Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure


...
InterruptWindowExiting0...1
bitValueType
UseTSCOffsetting0...1
bitValueType
HLTExiting0...1
bitValueType
INVLPGExiting0...1
bitValueType
MWAITExiting0...1
bitValueType
RDPMCExiting0...1
bitValueType
RDTSCExiting0...1
bitValueType
CR3LoadExiting0...1
bitValueType
CR3StoreExiting0...1
bitValueType
CR8LoadExiting0...1
bitValueType
CR8StoreExiting0...1
bitValueType
UseTPRShadow0...1
bitValueType
NMIWindowExiting0...1
bitValueType
MOVDRExiting0...1
bitValueType
UnconditionalIOExiting0...1
bitValueType
UseIOBitmaps0...1
bitValueType
MonitorTrapFlag0...1
bitValueType
UseMSRBitmaps0...1
bitValueType
MONITORExiting0...1
bitValueType
PAUSEExiting0...1
bitValueType
Activate2ndaryControls0...1
bitValueType

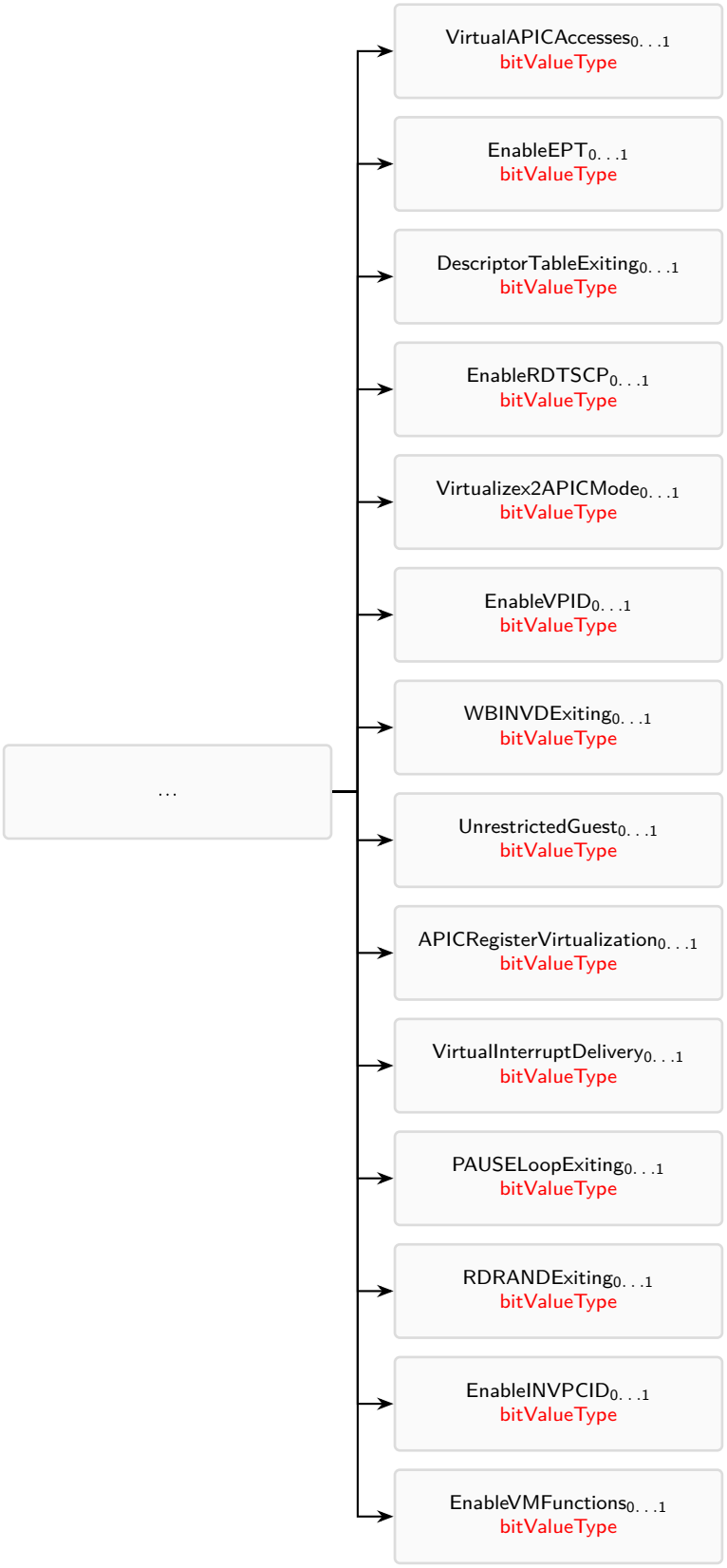
7.1.81 proc2CtrlType

The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions. These are the *primary processor-based* VM-execution controls and the *secondary processor-based* VM-execution controls.

The `proc2` element configures the secondary processor-based VM-execution controls, see Intel SDM Vol. 3C, "24.6.2 Processor-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.

Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure



7.1.82 vmEntryCtrlType

Configures Intel VMX VM-entry controls. These controls constitute a 32-bit vector that governs the basic operation of VM entries.

See Intel SDM Vol. 3C, "24.8.1 VM-Entry Controls" for more details and the meaning of the different bit-fields.

> ☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure

7.1.83 vmExitCtrlType

Configures Intel VMX VM-exit controls. These controls constitute a 32-bit vector that governs the basic operation of VM exits.

See Intel SDM Vol. 3C, "24.7.1 VM-Exit Controls" for more details and the meaning of the different bit-fields.

> ☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure

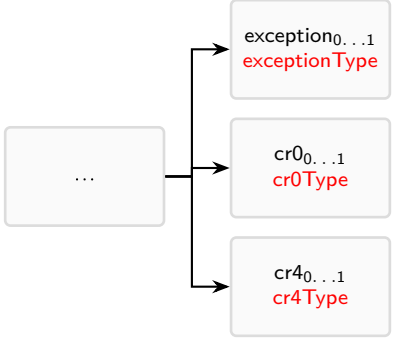
7.1.84 masksType

The `masks` element configures the Intel VMX CR0/CR4 guest/host masks and the guest/host exception bitmap.

In general, bits set to 1 in a guest/host mask correspond to bits *owned* by the host, causing a VM exit if the associated event occurs.

Reading from host owned bits in CR0/CR4 does not result in a VM exit but the value of the CR0/CR4 read shadow is returned instead (see Intel SDM Vol. 3C, "24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4").

Structure

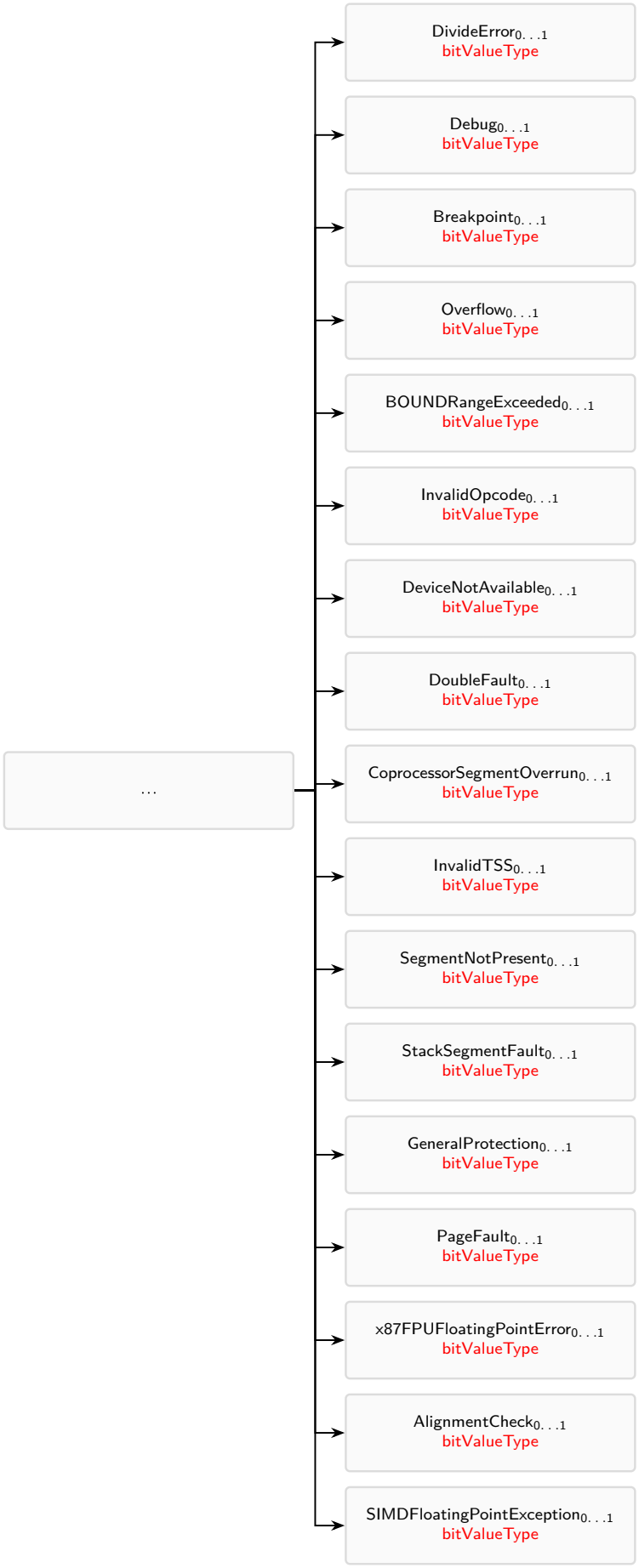


7.1.85 exceptionType

Configures Intel VMX exception bitmap. The exception bitmap is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exceptions vector.

See Intel SDM Vol. 3C, "24.6.3 Exception Bitmap" for more details on the exception bitmap configuration.

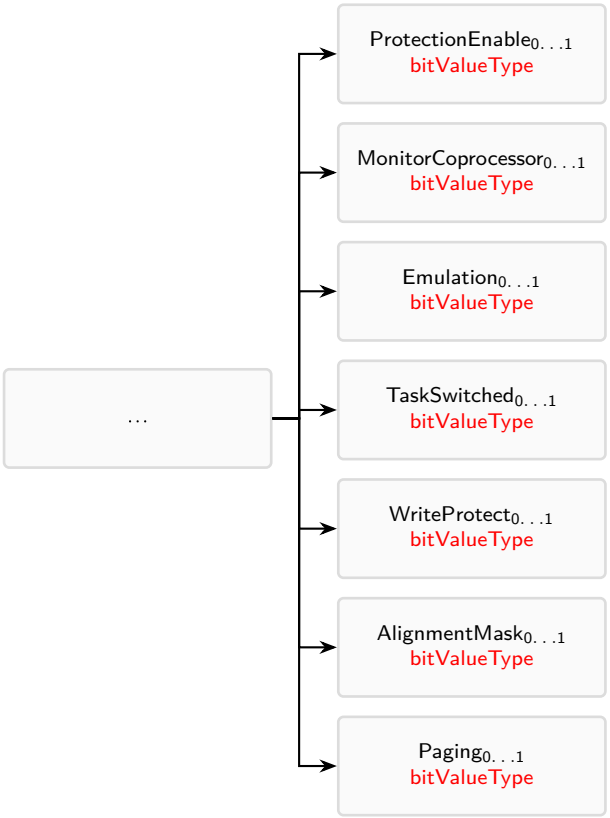
Structure


...
DivideError0...1
bitValueType
Debug0...1
bitValueType
Breakpoint0...1
bitValueType
Overflow0...1
bitValueType
BOUNDRangeExceeded0...1
bitValueType
InvalidOpcode0...1
bitValueType
DeviceNotAvailable0...1
bitValueType
DoubleFault0...1
bitValueType
CoprocessorSegmentOverrun0...1
bitValueType
InvalidTSS0...1
bitValueType
SegmentNotPresent0...1
bitValueType
StackSegmentFault0...1
bitValueType
GeneralProtection0...1
bitValueType
PageFault0...1
bitValueType
x87FPUFloatingPointError0...1
bitValueType
AlignmentCheck0...1
bitValueType
SIMDFloatingPointException0...1
bitValueType

7.1.86 cr0Type

Allows to set initial values of the CR0 control register or bits in the CR0 guest/host ownership mask.

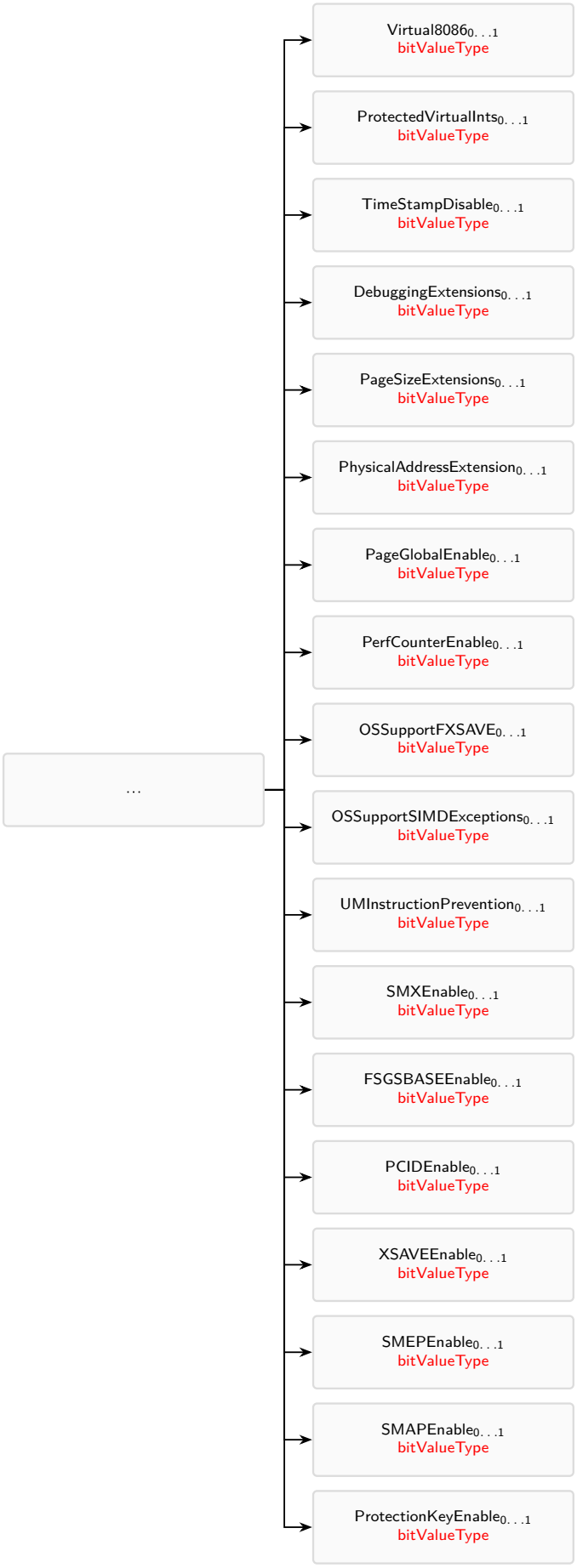
Structure



7.1.87 cr4Type

Allows to set initial values of the CR4 control register or bits in the CR4 guest/host ownership mask.

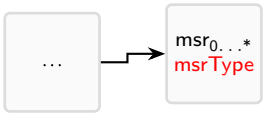
Structure


...
Virtual8086$_{0...1}$
bitValueType
ProtectedVirtualInts$_{0...1}$
bitValueType
TimeStampDisable$_{0...1}$
bitValueType
DebuggingExtensions$_{0...1}$
bitValueType
PageSizeExtensions$_{0...1}$
bitValueType
PhysicalAddressExtension$_{0...1}$
bitValueType
PageGlobalEnable$_{0...1}$
bitValueType
PerfCounterEnable$_{0...1}$
bitValueType
OSSupportFXSAVE$_{0...1}$
bitValueType
OSSupportSIMDExceptions$_{0...1}$
bitValueType
UMInstructionPrevention$_{0...1}$
bitValueType
SMXEnable$_{0...1}$
bitValueType
FSGSBASEEnable$_{0...1}$
bitValueType
PCIDEnable$_{0...1}$
bitValueType
XSAVEEnable$_{0...1}$
bitValueType
SMEPEnable$_{0...1}$
bitValueType
SMAPEnable$_{0...1}$
bitValueType
ProtectionKeyEnable$_{0...1}$
bitValueType

7.1.88 msrsType

List of model-specific registers (MSRs) a subject is allowed to access. The settings in this section are translated to the MSR bitmaps of the associated subject (as described by Intel SDM Vol. 3C, "24.6.9 MSR-Bitmap Address").

Structure



7.1.89 msrType

An `msr` element allows a subject direct access to the specified model-specific register (MSR).

> ☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator granting direct access to MSRs must be aware of the potential side-effects.

Attributes

Name	Type	Use
start MSR start address.	msrAddressType	required
end MSR end address.	msrAddressType	required
mode MSR access permissions.	msrModeType	required

7.1.90 msrAddressType

Base: xs:string

Start/end address value for MSRs in the low or high range:

- Low : `16#0000_0000# .. 16#0000_1fff#`
- High : `16#C000_0000# .. 16#C000_1fff#`

See also Intel SDM Vol. 3C, "24.6.9 MSR-Bitmap Address".

Restrictions

Pattern = 16#([cC0]000_)?[01]([0-9a-fA-F]3)#

7.1.91 msrModeType

Base: xs:string

MSR access rights.

Restrictions

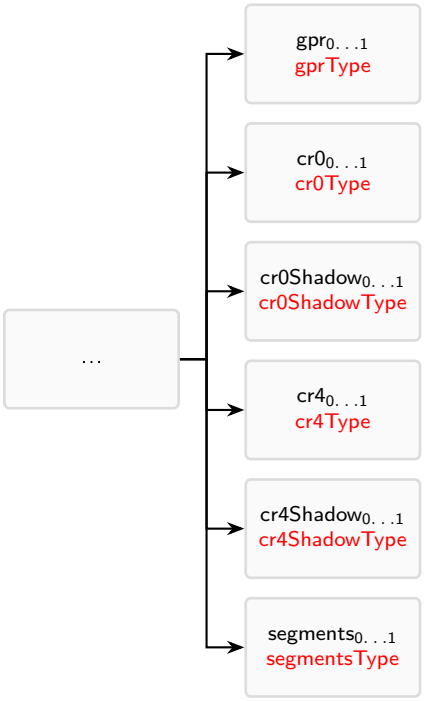
values:

- r
- w
- rw

7.1.92 registersType

The `registers` element specifies the initial value of general-purpose (GPR), CR0/CR4, CR0/CR4 read shadow and segment registers.

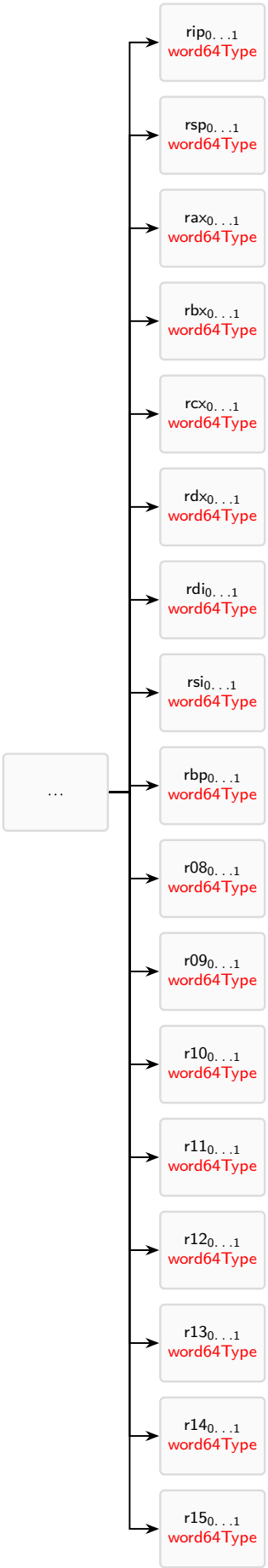
Structure



7.1.93 gprType

The `gpr` element specifies the initial values of subject general-purpose registers (GPRs).

Structure

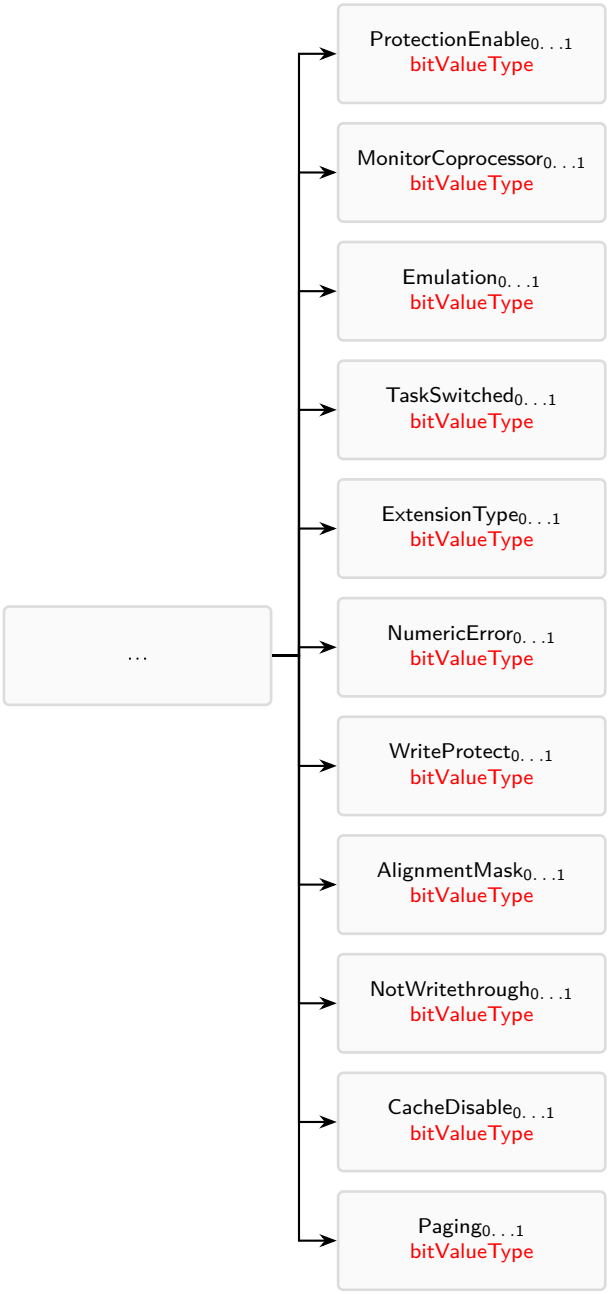

...
rip0...1
word64Type
rsp0...1
word64Type
rax0...1
word64Type
rbx0...1
word64Type
rcx0...1
word64Type
rdx0...1
word64Type
rdi0...1
word64Type
rsi0...1
word64Type
rbp0...1
word64Type
r080...1
word64Type
r090...1
word64Type
r100...1
word64Type
r110...1
word64Type
r120...1
word64Type
r130...1
word64Type
r140...1
word64Type
r150...1
word64Type

7.1.94 cr0ShadowType

Allows to set initial values of the CR0 shadow control register.

See Intel SDM Vol. 3C, "24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4" for more details on the CR0 shadow.

Structure

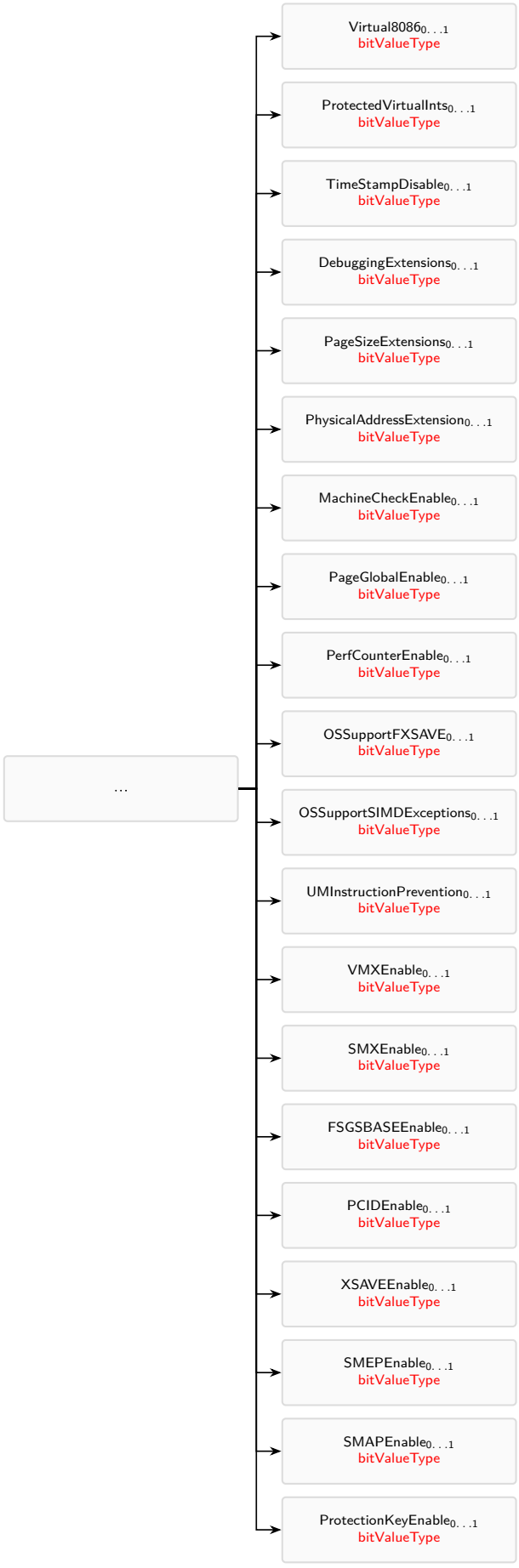


7.1.95 cr4ShadowType

Allows to set initial values of the CR4 shadow control register.

See Intel SDM Vol. 3C, "24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4" for more details on the CR0 shadow.

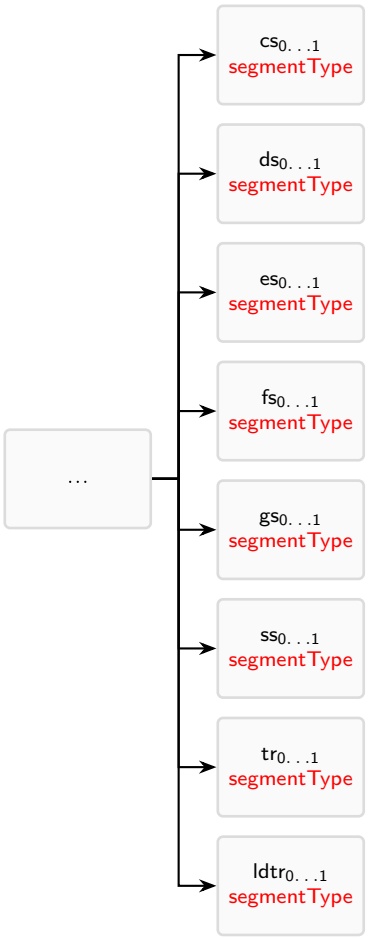
Structure


...
Virtual8086$_{0...1}$
bitValueType
ProtectedVirtualInts$_{0...1}$
bitValueType
TimeStampDisable$_{0...1}$
bitValueType
DebuggingExtensions$_{0...1}$
bitValueType
PageSizeExtensions$_{0...1}$
bitValueType
PhysicalAddressExtension$_{0...1}$
bitValueType
MachineCheckEnable$_{0...1}$
bitValueType
PageGlobalEnable$_{0...1}$
bitValueType
PerfCounterEnable$_{0...1}$
bitValueType
OSSupportFXSAVE$_{0...1}$
bitValueType
OSSupportSIMDExceptions$_{0...1}$
bitValueType
UMInstructionPrevention$_{0...1}$
bitValueType
VMXEnable$_{0...1}$
bitValueType
SMXEnable$_{0...1}$
bitValueType
FSGSBASEEnable$_{0...1}$
bitValueType
PCIDEnable$_{0...1}$
bitValueType
XSAVEEnable$_{0...1}$
bitValueType
SMEPEnable$_{0...1}$
bitValueType
SMAPEnable$_{0...1}$
bitValueType
ProtectionKeyEnable$_{0...1}$
bitValueType

7.1.96 segmentsType

The `segments` element specifies the initial values of subject segment registers.

Structure



7.1.97 segmentType

Initial value of a segment register, including hidden part. See Intel SDM Vol. 3A, "3.4.3 Segment Registers" for more details on segment registers.

Attributes

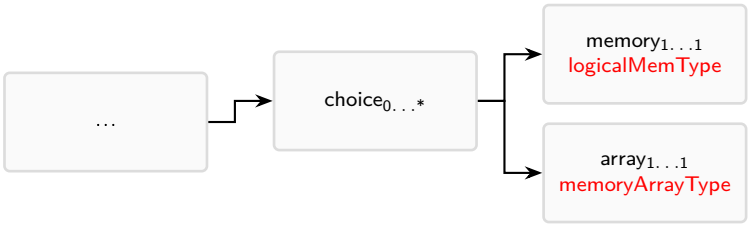
Name	Type	Use
selector Segment selector value.	word16Type	required
base Segment base address.	word64Type	required
limit Segment limit.	word32Type	required
access Segment access information.	word32Type	required

7.1.98 logicalMemoryType

In this section, components can specify expected memory mappings with given access rights and region size.

See line 948 in listing 8.1 for an example specification.

Structure



7.1.99 logicalMemType

The `memory` element requests a memory region with the specified `size` and permissions from the system. The region is expected to be placed at the address given via the `virtualAddress` attribute.

See line 953 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
size	word64Type	required
	Size of memory in bytes. Must be a multiple of page size (4K).	
virtualAddress	word64Type	required
	Expected address of memory mapping.	
logical	nameType	required
	Logical name of mapping.	
writable	booleanType	required
	Defines if the mapped memory is writable.	
executable	booleanType	required
	Defines if the memory region contents are executable by the processor.	

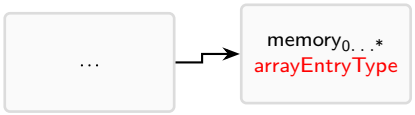
7.1.100 memoryArrayType

The memory array abstraction simplifies the declaration of consecutive memory mappings with a given base address, region size and `executable` and `writable` attributes. The child elements declare the number of expected regions.

Attributes

Name	Type	Use
logical	nameType	required
	Logical name of mapping.	
writable	booleanType	required
	Defines if the mapped memory is writable.	
executable	booleanType	required
	Defines if the memory region contents are executable by the processor.	
virtualAddressBase	word64Type	required
	Expected address of memory mapping.	
elementSize	word64Type	required
	Size of one array element in bytes. Must be a multiple of page size (4K).	

Structure



7.1.101 arrayEntryType

Array entries specify the number of array elements and assign a logical name to each element.

See line 1104 in listing 8.1 for an example array entry declaration.

Attributes

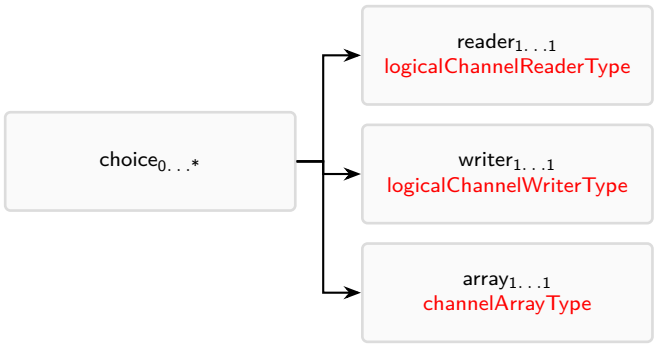
Name	Type	Use
logical	nameType	required
Logical name of array entry.		

7.1.102 logicalChannelsType

Components and libraries use the `channels` sub-section of `requires` to specify expected communication channels.

See line 969 in listing 8.1 for an example specification.

Structure



7.1.103 logicalChannelReaderType

The `reader` element requests a read-only channel of the specified size, address and optional notification vector.

See line 986 in listing 8.1 for an example channel reader specification.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of reader channel.		
virtualAddress	word64Type	required
Expected address of channel memory mapping.		
size	word64Type	required
Expected size of channel. Must be a multiple of page size (4K).		
vector	vectorType	optional
Notification vector.		

7.1.104 vectorType

Base: xs:nonNegativeInteger
Vector number.

Restrictions

value ≤255

7.1.105 logicalChannelWriterType

The `writer` element requests a channel with write permissions of the specified size, address and optional notification event number. For valid event ID ranges, see `vmcall` group in 7.1.137.
See line 974 in listing 8.1 for an example channel writer specification.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of writer channel.		
virtualAddress	word64Type	required
Expected address of channel memory mapping.		
size	word64Type	required
Expected size of channel. Must be a multiple of page size (4K).		
event	eventIdType	optional
Notification event number.		

7.1.106 eventIdType

Base: xs:nonNegativeInteger
Event number.

Restrictions

value ≤63

7.1.107 channelArrayType

The channel array abstraction simplifies the declaration of consecutive channel mappings with a given base address, channel size and optional event/vector bases. The child elements declare the number of expected channels and either the `reader` or `writer` role.
See line 1097 in listing 8.1 for an example specification.

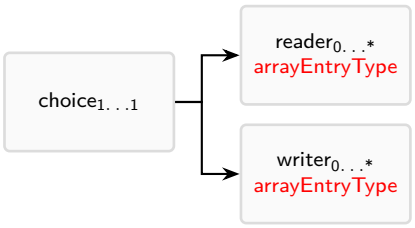
Attributes

Name	Type	Use
logical	nameType	required
Logical channel array name.		
eventBase	xs:nonNegativeInteger	optional
The eventBase attribute specifies the event number of the first element in the array. This number is incremented for all further elements in the array (eventBase + 1). For valid event ID ranges see vmcall group in 7.1.137. Note that this attribute is only taken into consideration for a writer array.		
vectorBase	vectorType	optional
The vectorBase attribute specifies the vector number of the first element in the array. This number is incremented for all further elements in the array (vectorBase + 1). Note that this attribute is only taken into consideration for a reader array.		

(continuation)

Name	Type	Use
virtualAddressBase	word64Type	required
Expected address of memory mapping.		
elementSize	word64Type	required
Size of one array element in bytes. Must be a multiple of page size (4K).		

Structure

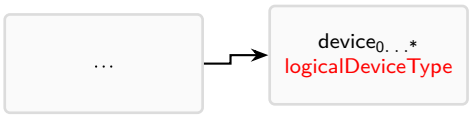


7.1.108 logicalDevicesType

The `devices` sub-section of the `requires` section is used to specify expected devices with their associated resources.

See line 1117 in listing 8.1 for an example specification.

Structure



7.1.109 logicalDeviceType

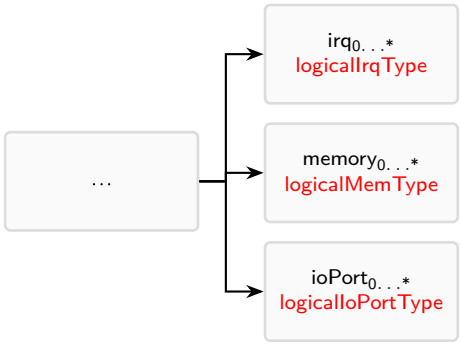
A `device` element specifies an expected logical device with its resources. Possible resources are `irq`, `memory` and `ioPort`.

See line 1122 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
logical	nameType	required
Logical device name.		

Structure



7.1.110 logicalIrqType

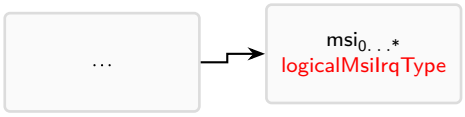
An `irq` element of a logical device reference requests an IRQ with given number from the system policy. The specified number will be injected when the device requires attention for the associated logical function.

See line 1127 in listing 8.1 for an example IRQ reference.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of IRQ resource.		
vector	vectorType	required
Expected IRQ number.		

Structure



7.1.111 logicalMsiIrqType

The presence of `msi` child elements of an `irq` device resource specifies that the component expects the device to be operated in MSI mode. The number of elements defines the expected MSI vector number count to be provided by the referenced device.

Attributes

Name	Type	Use
logical	nameType	required
Name of MSI resource.		

7.1.112 logicalIoPortType

The `ioPort` element requests a device I/O port resource with given range `start .. end` from the system.

See line 1294 in listing 8.1 for an example I/O port reference.

Attributes

Name	Type	Use
logical	nameType	required
Logical I/O port name.		
start	word16Type	required
I/O port start address.		
end	word16Type	required
I/O port end address.		

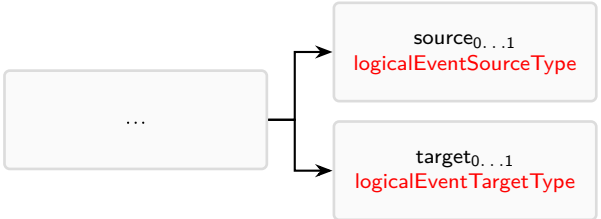
7.1.113 logicalEventsType

The `events` sub-section of the `requires` section is used to specify expected events with optional event actions.

A component can specify both source as well as target events.

See line 1211 in listing 8.1 for an example specification.

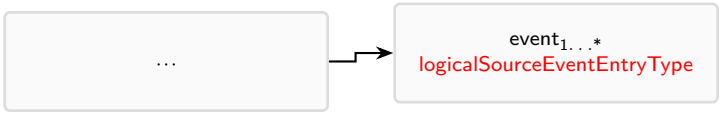
Structure



7.1.114 logicalEventSourceType

Specifies expected source events.

Structure



7.1.115 logicalSourceEventEntryType

Base: baseLogicalEventType
An entry in the component's source event list.

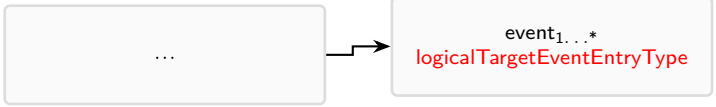
Attributes

Name	Type	Use
logical	nameType	required
Logical name of event.		
id	eventIdType	required
ID of source event. For valid ID ranges see vmcall group in 7.1.137.		

7.1.116 logicalEventTargetType

Specifies expected event targets.

Structure



7.1.117 logicalTargetEventEntryType

Base: baseLogicalEventType
An entry in the component's target event list.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of event.		
id	xs:nonNegativeInteger	optional
ID of target event entry.		

7.1.118 providedResourcesType

Components usually come in the form of an executable file. To this end, the `provides` section specifies the memory regions of the component binary executable with their content.

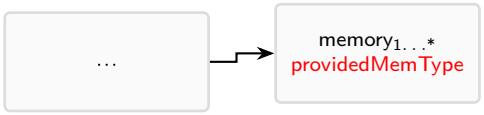
From a security perspective, it is often desirable to provide the different binary section as separate memory regions with the appropriate access rights, i.e. only the text section is executable, rodata is not writable and so on.

Memory specified in this sections are expanded to mapped physical regions for each subject that instantiates this component.

Note: the Mucbinsplit tool can be used to extract these section from an ELF binary into separate files and automatically add the corresponding memory elements to the component specification.

See line 1140 in listing 8.1 for an example `provides` section.

Structure



7.1.119 providedMemType

Base: memoryBaseType

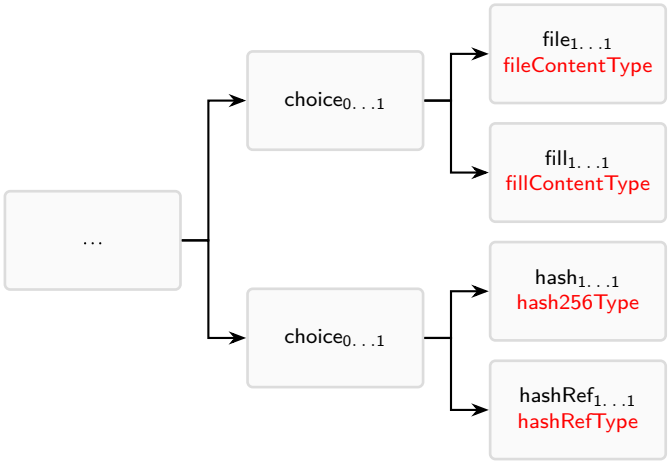
A `memory` element in the `provides` section declares memory region provided by the component. Mostly used to provide (a part) of the component binary.

See line 1158 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
size	memorySizeType	required
Size of region. Must be a multiple of page size (4K). Enforced by validator.		
virtualAddress	word64Type	required
Virtual address in component address space.		
type	subjectMemoryKindType	optional
Memory type (e.g. subject_binary).		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

Structure



7.1.120 subjectMemoryKindType

Base: memoryKindType < xs:string

Subject memory type to categorize memory assigned to a subject. The validator tool checks that a subject only maps memory regions of types outlined in this section (6.12).

Also used by build tools to lookup certain elements by type. For example, the `mugenzp` tool looks for subject memory of type `subject_zeropage` to process all Linux zero-pages in the policy.

The following memory types are currently supported:

- `subject`
  Generic subject memory, used e.g. for RAM regions of VM subjects. The `mugenzp` tool used for Linux VMs (5.3.11) exports such regions as `E820_RAM` in the ZP E820 memory map.

- `subject_info`
  Subject info (sinfo) region provided to all subjects. The sinfo region is used to query information about the execution environment. The file backing of this region is created by the `mugensinfo` tool (5.3.11).

- `subject_state`
  Subject execution state. Mapped into the SK kernel executing the given subject, kernels running on other CPUs have no access. Accessible by subject monitors running on the same CPU if specified in the policy. Validator enforces that each subject has an associated `subject_state` region and that it is mapped at the expected virtual address in the executing kernel (6.12).

- `subject_binary`
  Subject executable as a whole or separate subject executable regions (text, rodata, data, bss, stack) with access rights (writable/executable). The `mucbinsplit` tool automatically creates a component `provides` section with separate binary regions and associated backing files from a component binary (5.5.4).

- `subject_channel`
  Physical memory region used as shared channel between two subjects. The expander tool transforms channels in system policy source format to memory regions with this type in system policy format A/B, as described in section 7.1.69.

- `subject_crash_audit`
  Memory region used by crash audit facility to store system crash information into slots, see [2]. This information is preserved after a crash by performing a system warm start. Validators enforce that

  – Region is present and uncached, 6.8

  – Region does not overlap with image, 6.8

  - Kernel mappings are present and correct, 6.7
  - No subject has write access to this region, 6.12

- `subject_initrd`
  Physical memory of this type designates an *initial ramdisk*. This memory type is mostly used by Linux VMs. If multiple initrd regions are mapped into a subject, they must be adjacent (6.12).

  The `mugenzp` tool (5.3.11) extracts the virtual address and size of a subject-mapped region of this memory type and stores the values in the generated Linux zero-page (ZP) backing file.

- `subject_bios`
  Indicates to subjects that the memory region is reserved for BIOS/firmware and must not be used as regular RAM.

- `subject_acpi_*`
  Indicates to subjects that the memory region contains an ACPI table. See the ACPI specification for more information about RSDP, XSDT, FADT and DSDT ACPI tables. The `mugenzp` tool (5.3.11) exports such regions as `E820_ACPI` in the ZP E820 memory map.

- `subject_zeropage`
  Indicates to Linux subjects that the memory region contains a zero-page. See the Linux kernel Zero Page documentation for more information.

- `subject_solo5_boot_info`
  Indicates to a VM running Solo5/Mirage that the memory region contains a boot info structure. The file-backing of such a region may be created using the `mugensolo5` tool (5.3.11).

- `subject_device`
  Designates a memory region which is allowed to be added to a subject and a device domain. The difference to the `subject` memory type is that the region is not exported as `E820_RAM` but `E820_RESERVED` to Linux subjects. Therefore, such a region is useful to implement custom drivers without interference from Linux DMA zone handling.

- `subject_timed_event`
  Region designates a subject timed event page, as described in [1]. The expander tool creates a physical memory region for each subject and maps it into the associated subject and the SK kernel executing this subject.

Restrictions

values:

- subject
- subject_info
- subject_state
- subject_binary
- subject_channel
- subject_crash_audit
- subject_initrd
- subject_bios
- subject_acpi_rsdp
- subject_acpi_xsdt
- subject_acpi_fadt

- subject_acpi_dsdt
- subject_zeropage
- subject_solo5_boot_info
- subject_device
- subject_timed_event

7.1.121 componentType

Base: libraryType

A component is a piece of software which shall be executed by the SK. Components represent the building blocks of a component-based system and can be regarded as templates for executable entities instantiated by subjects.

The specification of a component declares the *binary program* by means of (file-backed memory) regions. It also specifies the component's view of the expected execution environment. A component may request the following resources from the system:

- Logical channels
- Logical memory regions
- Logical devices
- Logical events

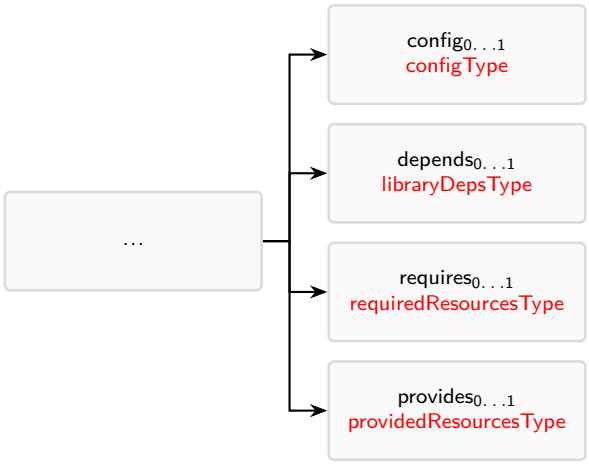
Components are identified by name and specify a profile. The profile controls the settings of the virtual CPU (vCPU).

See line 1044 in listing 8.1 for an example component.

Attributes

Name	Type	Use
name Component/library name.	nameType	required
profile Component profile.	componentProfileType	required

Structure



7.1.122 componentProfileType

Base: xs:string

The component profile defines default vCPU settings and triggers profile specific actions in the expander tool. The following actions are performed for the 'linux' profile.

- Add Linux zero-page (ZP, generated by Mugenzp)
- Add ACPI table regions (generated by Mugenacpi)
- Append sinfo address to boot parameters (`muen_sinfo`)
- Add dummy legacy BIOS regions (start address `16000c_0000`)
- Invalidate guest state of Linux SMP emulation sibling subjects

Restrictions

values:

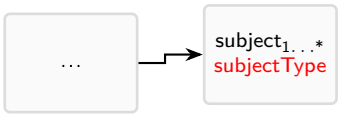
- native
- vm
- linux

7.1.123 subjectsType

The `subjects` element holds a list of subjects.

See line 1854 in listing 8.1 for an example subjects section.

Structure



7.1.124 subjectType

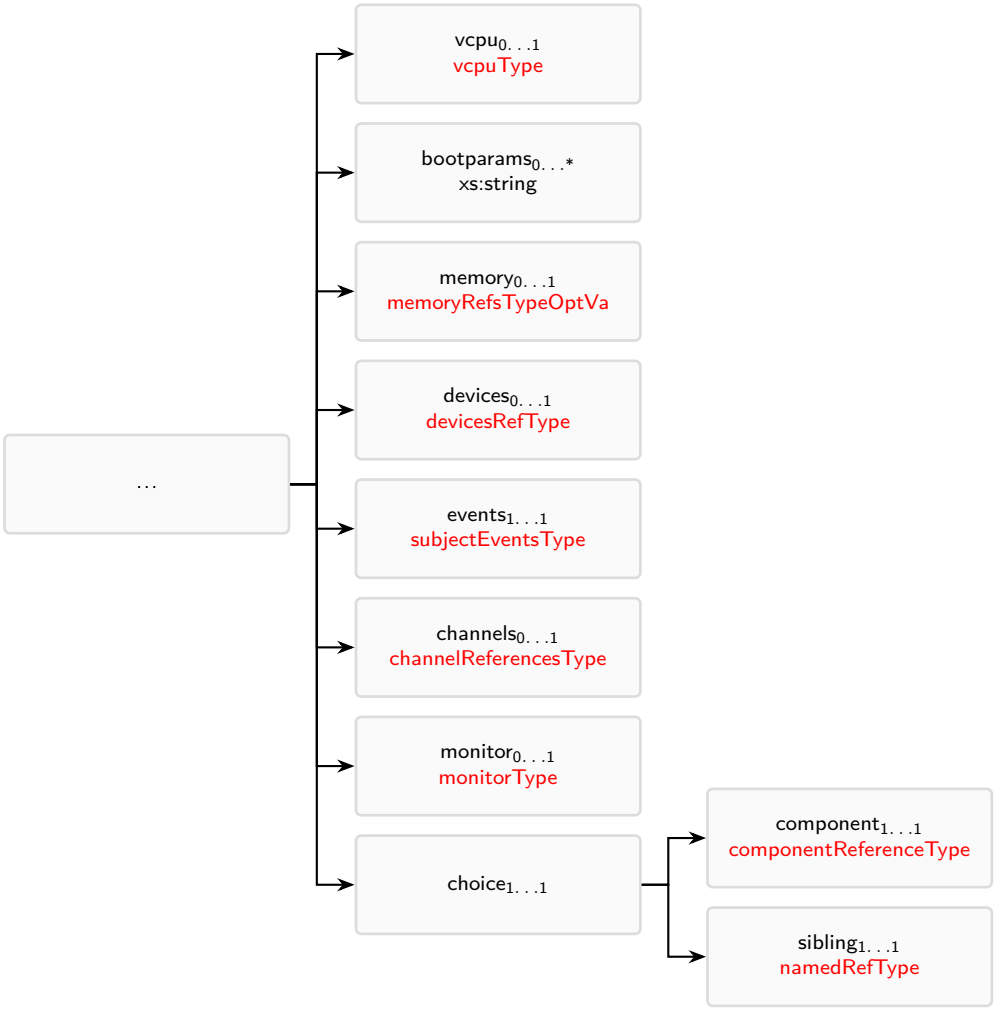
A subject is an instance of a component, i.e. an active component in the system policy that may be scheduled. Its specification references a component and maps all requested logical resources to physical resources provided by the system. Additional resources to the ones requested by the component can be specified here. This enables specialization of the base component specification.

See line 1858 in listing 8.1 for an example subject declaration.

Attributes

Name	Type	Use
name Unique subject name.	nameType	required

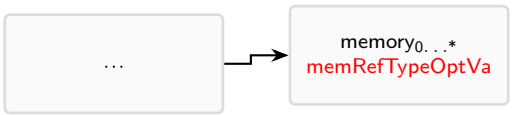
Structure



7.1.125 memoryRefsTypeOptVa

List of physical memory region references where 'virtualAddress' is optional.

Structure



7.1.126 memRefTypeOptVa

A `memory` element maps a physical memory region into the address space of a device domain or subject entity. The region will be accessible to the entity at the specified `virtualAdress` with permissions defined by the `executable` and `writable` attributes. If `virtualAddress` is omitted, it will be automatically generated by `mucfgvresalloc`.

Attributes

Name	Type	Use
`virtualAddress`	`word64Type`	`optional`
`Address in entity address space where the physical memory region is mapped.`		

```
(continuation)
Name              Type          Use

physical          nameType      required
  Name of referenced physical memory region.

logical           nameType      required
  Logical name of mapping.

writable          booleanType   required
  Defines if the mapped memory is writable.

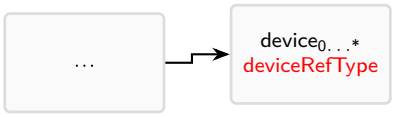
executable        booleanType   required
  Defines if the memory region contents are executable by the processor.
```

7.1.127 devicesRefType

List of device references. Used to grant a subject access to hardware devices and their resources.

See line 2257 in listing 8.1 for example device references.

Structure



7.1.128 deviceRefType

The `device` element allows a subject access to devices referenced via the `physical` attribute.

For PCI devices only a single virtual bus is provided (bus 0). The `pci` element may be used to place the device at a specific location (BDF). If no other logical device resources of the device are specified, then the expander tool will map all physical devices resources into the subject. When logical device resources are explicitly specified, then only access to those are actually granted. The physical attribute must be either a reference to an existing physical device, device alias or device class. Validators check that this is the case.

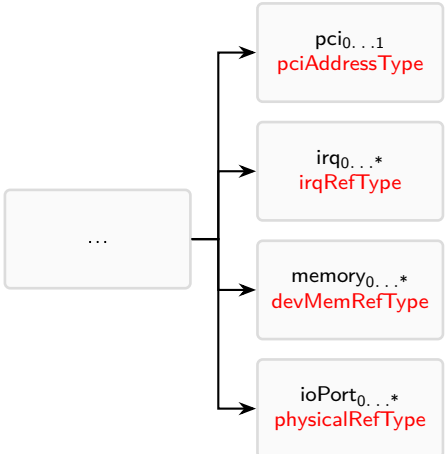
See line 2262 in listing 8.1 for an example reference.

Attributes

```
Name              Type          Use
-------------------------------------------------------------------------------
logical           nameType      required
  Logical device name.

physical          nameType      required
  Name of physical device to reference.
```

Structure



7.1.129 pciAddressType

PCI Bus, Device, Function triplet (BDF).

Attributes

Name	Type	Use
bus	byteType	required
PCI Bus number.		
device	pciDeviceNumberType	required
PCI Device number.		
function	pciFunctionNumberType	required
PCI Function number.		

7.1.130 irqRefType

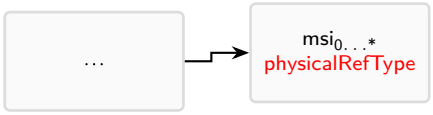
The device `irq` element assigns the referenced physical IRQ to the subject, i.e. if the device triggers the referenced physical IRQ, the specified `vector` number will be injected into the subject by the SK.

The presence of `msi` sub-elements enforces MSI mode (the default for MSI-capable devices and automatic device resource expansion).

Attributes

Name	Type	Use
logical	nameType	required
Logical IRQ name.		
physical	nameType	required
Name of physical device IRQ.		
vector	vectorType	optional
Vector to inject into subject if device triggers IRQ. Will be allocated by the expander if none is specified.		

Structure



7.1.131 physicalRefType

References a physical resource given by the `physical` attribute, and assigns a `logical` name to it.

Attributes

Name	Type	Use
logical	nameType	required
Logical name for resource reference.		
physical	nameType	required
Name of physical resource.		

7.1.132 devMemRefType

The device `memory` element maps the device memory region referenced via the `physical` attribute into the subject address space at address `virtualAddress`. The `executable`, `writable` attributes define the access permissions for the subject.

See line 2277 in listing 8.1 for an example device memory reference.

Attributes

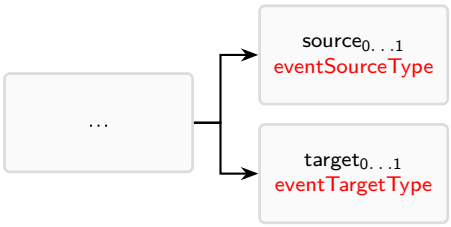
Name	Type	Use
virtualAddress	word64Type	optional
Address of mapping in subject address space. If none is specified, an identity mapping is applied by the expander tool.		
physical	nameType	required
Name of referenced physical memory region.		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

7.1.133 subjectEventsType

The subject `events` element specifies all events originating from or directed at this subject. The physical attribute is the name of a event defined in the global events section.

See line 1877 in listing 8.1 for an example subject events section.

Structure



7.1.134 eventSourceType

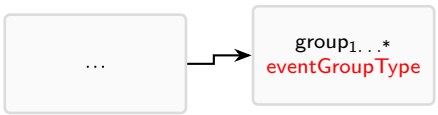
The event `source` element specifies events that are allowed to be triggered by the associated subject.

Source events are divided into two groups: `vmx_exit` and `vmcall`. For event group `vmx_exit` the id attribute specifies the trap number while in the `vmcall` group it designates the hypercall number. For the valid range of IDs for each group see section 7.1.137.

The `vmx_exit` group is translated to a lookup table for handling VMX exit traps as defined by Intel SDM Vol. 3D, "Appendix C VMX Basic Exit Reasons". The `vmcall` group on the other hand is translated into a lookup table to handle hypercalls.

See line 1883 in listing 8.1 for an example event source section.

Structure



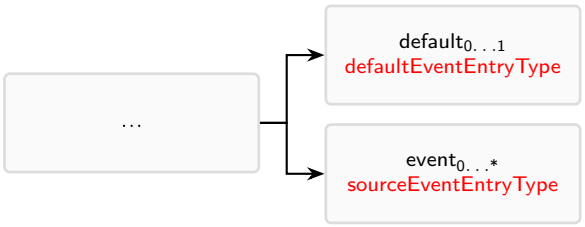
7.1.135 eventGroupType

Source event group element. Currently, two groups are supported: `vmcall` for hypercalls and `vmx_exit` for all other supported traps.

Attributes

Name	Type	Use
name	eventGroupNameType	required
Name of event group.		

Structure



7.1.136 defaultEventEntryType

Base: baseDefaultEventType

The `default` element entry can be used to specify an event which should be added for all event ids that have not been explicitly specified.

See line 1894 in listing 8.1 for a default source event example.

Attributes

Name	Type	Use
physical	nameType	required
Global event reference.		

7.1.137 sourceEventEntryType

Base: baseEventWithIDType < baseEventType

A source `event` entry specifies a source event node, i.e. it registers a handler for the given event `id`. These IDs, depending on the event group, are either hypercall numbers or VMX basic exit reasons. The valid ID ranges of the respective groups are:

vmx_exit 0 .. 59

vmcall 0 .. 63

Additionally, the following IDs in `vmx_exit` group are reserved and may not be used:

- Used by kernel: 1, 7, 41, 52, 55
- Reserved by Intel: 35, 38, 42

It is possible to assign event actions to event source entries. Currently supported source event actions are `subject_sleep`, `subject_yield`, `unmask_irq`, `system_reboot`, `system_poweroff` and `system_panic`, which all have the kernel itself as endpoint.

See line 1951 in listing 8.1 for a source event entry example.

(continuation)

Name	Type	Use

Attributes

Name	Type	Use
logical Logical event name.	nameType	required
physical Physical event name.	nameType	required
id ID of event.	eventIdType	required

7.1.138 eventGroupNameType

Base: xs:string

Supported event groups.

Restrictions

values:

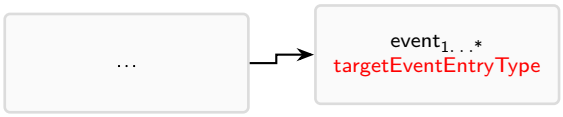
- vmx_exit
- vmcall

7.1.139 eventTargetType

The event `target` element specifies events that the subject is an *endpoint* of.

See line 1981 in listing 8.1 for an example event target section.

Structure



7.1.140 targetEventEntryType

Base: baseEventType

The `event` element in the target section specifies one event endpoint by referencing a physical event and assigning a logical name to it.

See line 1986 in listing 8.1 for an example event endpoint.

Attributes

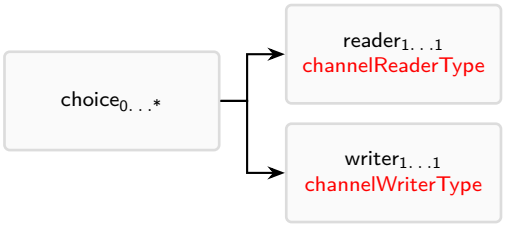
Name	Type	Use
logical Logical event name.	nameType	required
physical Physical event name.	nameType	required
id Event ID.	xs:nonNegativeInteger	optional

7.1.141 channelReferencesType

The `channel` section of a subject declares references to communication channels. The referenced channels become accessible to the requesting subject either as reader or writer endpoint.

See line 2389 in listing 8.1 for an example section.

Structure



7.1.142 channelReaderType

A channel `reader` element references a global communication channel as reader endpoint, i.e. the channel is mapped read-only into the subject address space.

See line 2395 in listing 8.1 for an example reader declaration.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of reader channel.		
physical	nameType	required
Name of physical channel.		
virtualAddress	word64Type	optional
Address of mapping in subject address space.		
vector	vectorOrAutoType	optional
Associated vector. Must be set if a physical channel with hasEvent mode != switch is referenced (enforced by validator). The vector attribute is optional in the case of mode switch.		

7.1.143 vectorOrAutoType

Vector number or "auto" to request automatic assignment.

Restrictions

Union of

- vectorType
- constantAuto

7.1.144 channelWriterType

A channel `writer` element references a global communication channel as writer endpoint, i.e. the channel is mapped with write permissions into the subject address space.

See line 2402 in listing 8.1 for an example writer declaration.

```
(continuation)
Name              Type                Use
```

Attributes

```
Name              Type                Use
-------------------------------------------------------------------------------
logical           nameType            required
 Logical name of writer channel.

physical          nameType            required
 Name of physical channel.

virtualAddress    word64Type          optional
 Address of mapping in subject address space.

event             eventIdOrAutoType   optional
 Associated event number.  Must be set if a physical channel with hasEvent
 attribute is referenced.
```

7.1.145 eventIdOrAutoType

Event number or "auto" to request automatic assignment.

Restrictions

Union of

- eventIdType
- constantAuto

7.1.146 monitorType

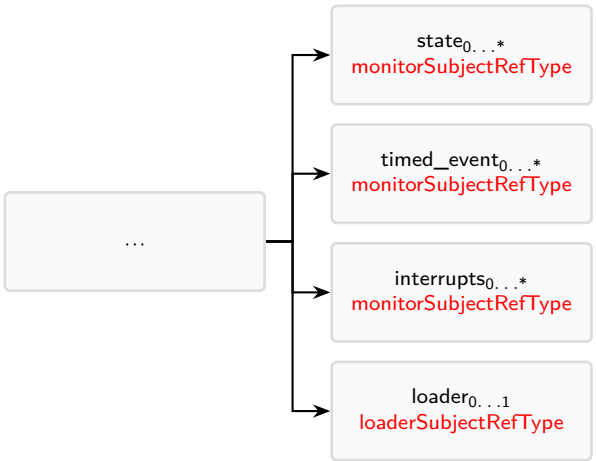
The monitor abstraction enables subjects to request access to certain data of another subject specified by name. Possible child elements are:

- State
- Timed_Events
- Interrupts
- Loader

See the Muen Component Specification document for details about these subject monitor interfaces.

See line 1997 in listing 8.1 for an example monitor section.

Structure



7.1.147 monitorSubjectRefType

Base: loaderSubjectRefType

Give subject monitor (SM) access to the referenced subject state.

Attributes

Name	Type	Use
subject	nameType	required
Name of monitored subject.		
logical	nameType	required
Logical name of state mapping.		
virtualAddress	word64Type	required
Address to map requested subject address space.		
writable	booleanType	required
Whether or not the given state is mapped writable into the SM.		

7.1.148 loaderSubjectRefType

The `loader` mechanism effectively puts the loaded subject denoted by the `subject` attribute under loader control, as it is not able to start without the help of the loader.

In more detail, the `loader` monitor element instructs the expander tool to map all memory regions of the referenced subject into the address space of the monitor subject, using the specified `virtualAddress` as offset in the address space of the loader.

If a memory region of the loaded subject is writable and file-backed, the region is replaced with an empty region and linked via the `hashRef` mechanism to the original region which is mapped into the loader.

The state of the loaded subject is then invalidated by clearing the `CR4.VMXE` bit in the initial subject CR4 register value. If such a subject is scheduled by the kernel, a VMX exit *VM-entry failure due to invalid guest state* (33) occurs. See Intel SDM Vol. 3C, "23.7 Enabling and Entering VMX Operation" and Intel SDM Vol. 3C, "23.8 Restrictions on VMX Operation" for more details. This trap is linked to the loader via normal VMX event handling. After handover, the loader initializes the memory regions replaced by the expander with the designated content.

All information required to *load* the loaded subject is provided to the loader subject via its own sinfo API. Memory regions prefixed with `monitor_sinfo_` provide access to the sinfo regions of the loaded subjects. Regions prefixed with `monitor_state_` specify memory regions containing the subject register state of the loaded subject.

The difference between the `monitor_sinfo_` memory region address in the loader and the address of the `sinfo` memory region in the target sinfo information denotes the `virtualAddress` offset attribute of the `loader` element in the policy. This information combined is enough to fully construct the initial state of the loaded subject, or to reset a subject to its initial state on demand.

The loader may also optionally check the hashes of the restored regions, as this information is provided via the sinfo mechanism as well.

See line 2014 in listing 8.1 for an example loader element.

Attributes

Name	Type	Use
subject	nameType	required
Name of monitored subject.		
logical	nameType	required
Logical name of state mapping.		
virtualAddress	word64Type	required
Address to map requested subject address space.		

7.1.149 componentReferenceType

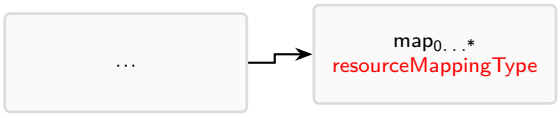
The `component` reference element specifies which component this subject instantiates. All logical resources required by the component must be mapped to physical resources of the appropriate type. Validators make sure that all requirements are satisfied and that no mapping has been omitted.

See line 1908 in listing 8.1 for an example component reference.

Attributes

Name	Type	Use
ref	nameType	required
Name of referenced component.		

Structure



7.1.150 resourceMappingType

The `map` element maps a physical resource provided by the system with a resource requested by the referenced component.

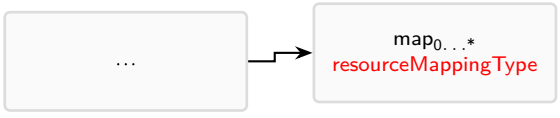
This element allows recursion to map child resources as well (e.g. device memory, I/O ports etc).

See line 1924 in listing 8.1 for an example mapping.

Attributes

Name	Type	Use
logical	nameType	required
Name of logical resource requested by the component.		
physical	nameType	required
Physical name of resource.		

Structure



7.1.151 schedulingType

The Muen SK implements a fixed, cyclic scheduler. The `scheduling` element is used to specify such a static plan by means of a major frame. A major frame consist of an arbitrary number of minor frames. Minor frames in turn specify a duration in number of ticks a scheduling partition is scheduled.

Scheduling partitions defined in the `partitions` element consist of one or more scheduling groups, which in turn specify one or more subjects to be scheduled. *Scheduling groups* are used to define groups of cooperating subjects, which are allowed to hand over execution to a subject in the same scheduling group. This is done via *handover* events. Membership of a scheduling group must be specified explicitly in the policy, validators enforce that these settings are correct by calculating the chain of handover events.

While scheduling groups support the efficient cooperation of multiple subjects, subjects which need to be spatially but not temporally isolated from each other cannot profit from it. To efficiently support this use-case, the scheduling partition concept is implemented.

Within a *scheduling partition*, all scheduling groups are scheduled round robin with preemption and the opportunity to yield and/or sleep. If a subject in a scheduling group sleeps or yields, the next scheduling group in the scheduling partition is scheduled. More precisely: the active subject of the next scheduling group is executed by the SK.

Note that prioritization is not implemented on purpose to avoid any starvation issues[2]. The yield operation maps to the x86_64 `PAUSE` instruction, while sleep corresponds to `HLT`. See the *Muen Component Specification* document [1] for more information on this topic.

Minor frames designate the scheduling partition that is to be executed for the given amount of ticks. The scheduling partition attribute `name` uniquely identifies a scheduling partition. On first activation, the first scheduling group (in XML-order) is scheduled. Within the scheduling group, the first subject (again in XML-order) is executed. The active subject of a scheduling group may change over time, as the cooperating subjects initiate handover events.

The tickRate attribute of the `scheduling` element has the unit Hertz (Hz) and specifies the number of clock ticks per second. The ticks attribute of minor frames is expressed in terms of this tick rate. As an example: if we want to declare the minor frame duration in terms of microseconds (10^{-6}) then a tick rate of 1000000 must be used.

The duration of a major frame must be the same on each CPU, meaning the sum of all minor frame ticks for any given CPU must be identical. However, different major frames can have arbitrary length.

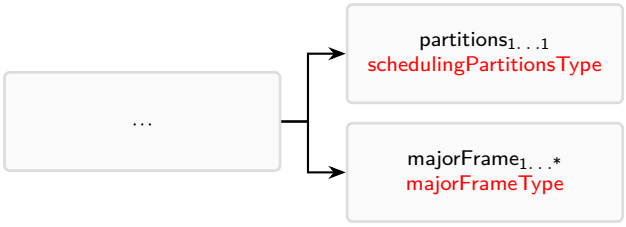
The Tau0 subject designates to the kernel which major frame is the currently active one. At the end of each major frame, the kernel determines the active major frame and switches to that scheduling plan for the duration of the major frame.

See line 2465 in listing 8.1 for an example scheduling plan.

Attributes

Name	Type	Use
tickRate	xs:positiveInteger	required
Scheduling clock ticks in Hz.		

Structure

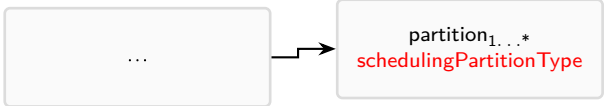


7.1.152 schedulingPartitionsType

The `partitions` element is used to specify all scheduling partitions of the system.

See line 2524 in listing 8.1 for an example partitions element.

Structure



7.1.153 schedulingPartitionType

Base: baseSchedulingPartitionType

The scheduling `partition` element is used to specify a collection of scheduling groups consisting of subjects that require spatial but not temporal isolation from each other. Within a scheduling

[2]Prioritization with starvation protection cannot be implemented with low complexity

partition, all scheduling groups are scheduled round robin with preemption (i.e. non-cooperatively) and the opportunity to yield and/or sleep.

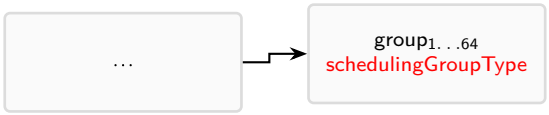
A scheduling partition must contain at least one scheduling group.

See line 2529 in listing 8.1 for an example scheduling partition.

Attributes

Name	Type	Use
name	nameType	required
Name of the scheduling partition.		

Structure



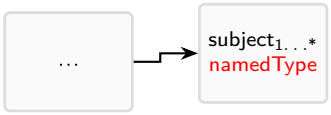
7.1.154 schedulingGroupType

Base: baseSchedulingGroupType

The scheduling `group` element is used to specify a collection of subjects that may cooperatively schedule each other via handover events. Scheduling groups must contain at least one subject. As an example, a Linux subject and its associated Subject Monitor (SM), Subject Loader (SL) and Device Manager (DM) form a scheduling group.

See line 2539 in listing 8.1 for an example scheduling group.

Structure



7.1.155 namedType

The `namedType` is used for simple elements in the policy, that only specify a name.

Attributes

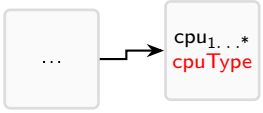
Name	Type	Use
name	nameType	required
Name of element.		

7.1.156 majorFrameType

A major frame consists of a sequence of minor frames for a given CPU. When the end of a major frame is reached, all CPUs synchronize and the scheduler starts over from the beginning using the first minor frame again. This means that major frames are repeated in a cyclic fashion until a different major frame is designated via the Tau0 interface.

See line 2602 in listing 8.1 for an example major frame.

Structure



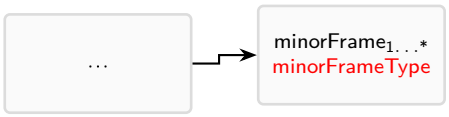
7.1.157 cpuType

The cpu element is used to specify major frames for each CPU of the system.
See line 2610 in listing 8.1 for an example cpu element.

Attributes

Name	Type	Use
id ID of CPU.	xs:nonNegativeInteger	required

Structure



7.1.158 minorFrameType

A minor frame specifies the number of scheduling ticks a partition is allowed to run on the CPU specified by the parent cpu element.
See line 2615 in listing 8.1 for an example minor frame.

Attributes

Name	Type	Use
partition Name of scheduled partition.	nameType	required
ticks Number of scheduling ticks in minor frame.	xs:positiveInteger	required

Chapter 8

Appendix

8.1 Annotated Example Policy

```
<?xml version='1.0' encoding='utf-8'?>
<system>
 <!--
  A Muen system policy specifies all hardware resources such as physical
  memory, devices, CPU time, etc and how these resources are accessed by
  the separation kernel, the subjects and devices.

  The `system` section is the top-level element in the Muen system policy.
  It contains various sub-elements which specify all aspects of a concrete
  system.

  This is the *source format* of the Muen system policy. It allows for
  abstractions, such as channels, which are broken down into their
  constituent parts by the toolchain in format A and B accordingly.
 -->
 <config>
  <!--
   The purpose of a config section is to specify configuration values which
   parameterize a system or a component. It allows to declare boolean,
   string and integer values. The following sections in the system policy
   provide support for configuration values:

   -   System

   -   Platform

   -   Component

   During the build process, configuration values provided by the platform
   are merged into the global system configuration. Component configuration
   values allow the parameterization of component-local functionality.

   Besides component parameterization, configuration options can be used in
   `if` conditionals, as shown in the following example.

   ``` xml
   <if variable="xhcidbg_enabled" value="true">
   ...
   </if>
   ```

   If the type of the referenced variable is string the comparison is
   case-sensitive. A second use case is XML attribute value expansion as
   follows:

   ``` xml
   <channel name="debuglog" size="$logchannel_size"/>
   ```

   The `size` attribute value is not specified directly, but parameterized
   via an integer configuration option.
  -->
   <boolean name="pciconf_emulation_enabled" value="true"/>
   <boolean name="pciconf_emulation_xhci_enabled" value="false"/>
   <boolean name="xhcidbg_supported" value="false"/>
   <boolean name="xhcidbg_enabled" value="true"/>
   <boolean name="dbgserver_sink_serial" value="true"/>
   <boolean name="ahci_supported" value="true"/>
   <boolean name="serial_supported" value="true"/>
   <boolean name="dbgserver_sink_xhcidbg" value="false"/>
   <boolean name="linux_debug" value="false"/>
```

```
    <boolean name="ahci_drv_enabled" value="false"/>
    <boolean name="dbgserver_sink_shmem" value="false"/>
    <boolean name="uefi_gop_rmrr_access" value="false"/>
    <boolean name="hsuart_supported" value="false"/>
    <boolean name="ahci_drv_active" value="false"/>
    <boolean name="dbgserver_sink_pcspkr" value="false"/>
    <boolean name="dbgserver_serial_enabled" value="true"/>
    <string name="pciconf_emulation_xhci_devid" value="16#02#"/>
    <string name="pciconf_emulation_nic_devid" value="16#01#"/>
    <string name="logchannel_size" value="16#0002_0000#"/>
    <string name="platform" value="platform/lenovo-t430s.xml"/>
    <string name="hardware" value="hardware/lenovo-t430s.xml"/>
    <string name="pciconf_emulation_xhci_physdev" value="usb_controller_1"/>
    <string name="system" value="xml/demo_system_vtd.xml"/>
    <string name="pciconf_emulation_nic_physdev" value="ethernet_controller_1"/>
    <string name="igd_opregion_address" value="16#baf5_5000#"/>
    <string name="additional_hardware" value="hardware/common_hardware.xml"/>
  </config>
  <hardware>
    <!--
     Systems running the Muen SK perform static resource allocation at
     integration time. This means that all available hardware resources of a
     target machine must be defined in the system policy in order for these
     resources to be allocated to subjects.

     The `hardware` element is the top-level element of the hardware
     specification in the system policy. Information provided by a hardware
     description includes the amount of available physical memory blocks
     including reserved memory regions (RMRR), the number of physical CPU
     cores and hardware device resources.

     The Muen toolchain provides a handy tool to automate the cumbersome
     process of gathering hardware resource data from a running Linux system:
     `mugenhwcfg`[^1].

     [^1]: https://git.codelabs.ch/?p=muen/mugenhwcfg.git
    -->
    <processor cpuCores="2" speed="2893431" vmxTimerRate="5">
      <!--
       The `processor` element specifies the number of CPU cores, the processor
       speed in kHz and the Intel VMX preemption timer rate.

       Since Intel CPUs can have arbitrary APIC identifiers, the APIC IDs of
       all physical CPUs are enumerated here. The APIC ID is required for
       interrupt and IPI routing.

       The `processor` element also lists register values for all CPUID leaves
       of the hardware target, and some MSR values of interest.
      -->
      <cpu apicId="0"/>
      <cpu apicId="2"/>
      <cpuid eax="16#0000_000d#" ebx="16#756e_6547#" ecx="16#6c65_746e#" edx="16#4965_6e69#" leaf="
       16#0000_0000#" subleaf="16#00#"/>
      <cpuid eax="16#0003_06a9#" ebx="16#0010_0800#" ecx="16#7fba_e3ff#" edx="16#bfeb_fbff#" leaf="
       16#0000_0001#" subleaf="16#00#"/>
      <cpuid eax="16#7603_5a01#" ebx="16#00f0_b2ff#" ecx="16#0000_0000#" edx="16#00ca_0000#" leaf="
       16#0000_0002#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
       16#0000_0003#" subleaf="16#00#"/>
      <cpuid eax="16#1c00_4121#" ebx="16#01c0_003f#" ecx="16#0000_003f#" edx="16#0000_0000#" leaf="
       16#0000_0004#" subleaf="16#00#"/>
      <cpuid eax="16#1c00_4122#" ebx="16#01c0_003f#" ecx="16#0000_003f#" edx="16#0000_0000#" leaf="
       16#0000_0004#" subleaf="16#01#"/>
      <cpuid eax="16#1c00_4143#" ebx="16#01c0_003f#" ecx="16#0000_01ff#" edx="16#0000_0000#" leaf="
       16#0000_0004#" subleaf="16#02#"/>
      <cpuid eax="16#1c03_c163#" ebx="16#03c0_003f#" ecx="16#0000_0fff#" edx="16#0000_0006#" leaf="
       16#0000_0004#" subleaf="16#03#"/>
      <cpuid eax="16#0000_0040#" ebx="16#0000_0040#" ecx="16#0000_0003#" edx="16#0002_1120#" leaf="
       16#0000_0005#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0077#" ebx="16#0000_0002#" ecx="16#0000_0009#" edx="16#0000_0000#" leaf="
       16#0000_0006#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0000#" ebx="16#0000_0281#" ecx="16#0000_0000#" edx="16#9c00_0400#" leaf="
       16#0000_0007#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
       16#0000_0008#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
       16#0000_0009#" subleaf="16#00#"/>
      <cpuid eax="16#0730_0403#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0603#" leaf="
       16#0000_000a#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0001#" ebx="16#0000_0002#" ecx="16#0000_0100#" edx="16#0000_0000#" leaf="
       16#0000_000b#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0004#" ebx="16#0000_0004#" ecx="16#0000_0201#" edx="16#0000_0000#" leaf="
       16#0000_000b#" subleaf="16#01#"/>
      <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
       16#0000_000c#" subleaf="16#00#"/>
      <cpuid eax="16#0000_0007#" ebx="16#0000_0340#" ecx="16#0000_0340#" edx="16#0000_0000#" leaf="
       16#0000_000d#" subleaf="16#00#"/>
```

```
    <cpuid eax="16#0000_0001#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
     16#0000_000d#" subleaf="16#01#"/>
    <cpuid eax="16#0000_0100#" ebx="16#0000_0240#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
     16#0000_000d#" subleaf="16#02#"/>
    <cpuid eax="16#0000_0007#" ebx="16#0000_0340#" ecx="16#0000_0340#" edx="16#0000_0000#" leaf="
     16#2000_0000#" subleaf="16#00#"/>
    <cpuid eax="16#8000_0008#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
     16#8000_0000#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0001#" edx="16#2810_0800#" leaf="
     16#8000_0001#" subleaf="16#00#"/>
    <cpuid eax="16#2020_2020#" ebx="16#4920_2020#" ecx="16#6c65_746e#" edx="16#2029_5228#" leaf="
     16#8000_0002#" subleaf="16#00#"/>
    <cpuid eax="16#6572_6f43#" ebx="16#294d_5428#" ecx="16#2d37_6920#" edx="16#3032_3533#" leaf="
     16#8000_0003#" subleaf="16#00#"/>
    <cpuid eax="16#5043_204d#" ebx="16#2040_2055#" ecx="16#3039_2e32#" edx="16#007a_4847#" leaf="
     16#8000_0004#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
     16#8000_0005#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0100_6040#" edx="16#0000_0000#" leaf="
     16#8000_0006#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0000#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0100#" leaf="
     16#8000_0007#" subleaf="16#00#"/>
    <cpuid eax="16#0000_3024#" ebx="16#0000_0000#" ecx="16#0000_0000#" edx="16#0000_0000#" leaf="
     16#8000_0008#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0007#" ebx="16#0000_0340#" ecx="16#0000_0340#" edx="16#0000_0000#" leaf="
     16#8086_0000#" subleaf="16#00#"/>
    <cpuid eax="16#0000_0007#" ebx="16#0000_0340#" ecx="16#0000_0340#" edx="16#0000_0000#" leaf="16#
     c000_0000#" subleaf="16#00#"/>
    <msr address="16#0000_003a#" name="IA32_FEATURE_CONTROL" regval="16#0000_0000_0000_0005#"/>
    <msr address="16#0000_01a0#" name="IA32_MISC_ENABLE" regval="16#0000_0000_0085_0089#"/>
    <msr address="16#0000_0480#" name="IA32_VMX_BASIC" regval="16#00da_0400_0000_0010#"/>
    <msr address="16#0000_0481#" name="IA32_VMX_PINBASED_CTLS" regval="16#0000_007f_0000_0016#"/>
    <msr address="16#0000_0482#" name="IA32_VMX_PROCBASED_CTLS" regval="16#fff9_fffe_0401_e172#"/>
    <msr address="16#0000_0483#" name="IA32_VMX_EXIT_CTLS" regval="16#007f_ffff_0003_6dff#"/>
    <msr address="16#0000_0484#" name="IA32_VMX_ENTRY_CTLS" regval="16#0000_ffff_0000_11ff#"/>
    <msr address="16#0000_0485#" name="IA32_VMX_MISC" regval="16#0000_0000_1004_01e5#"/>
    <msr address="16#0000_0486#" name="IA32_VMX_CR0_FIXED0" regval="16#0000_0000_8000_0021#"/>
    <msr address="16#0000_0487#" name="IA32_VMX_CR0_FIXED1" regval="16#0000_0000_ffff_ffff#"/>
    <msr address="16#0000_0488#" name="IA32_VMX_CR4_FIXED0" regval="16#0000_0000_0000_2000#"/>
    <msr address="16#0000_0489#" name="IA32_VMX_CR4_FIXED1" regval="16#0000_0000_0017_67ff#"/>
    <msr address="16#0000_048b#" name="IA32_VMX_PROCBASED_CTLS2" regval="16#0000_08ff_0000_0000#"/>
    <msr address="16#0000_048c#" name="IA32_VMX_EPT_VPID_CAP" regval="16#0000_0f01_0611_4141#"/>
    <msr address="16#0000_048d#" name="IA32_VMX_TRUE_PINBASED_CTLS" regval="16#0000_007f_0000_0016#"/
     >
    <msr address="16#0000_048e#" name="IA32_VMX_TRUE_PROCBASED_CTLS" regval="16#fff9_fffe_0400_6172#"
     />
    <msr address="16#0000_048f#" name="IA32_VMX_TRUE_EXIT_CTLS" regval="16#007f_ffff_0003_6dfb#"/>
    <msr address="16#0000_0490#" name="IA32_VMX_TRUE_ENTRY_CTLS" regval="16#0000_ffff_0000_11fb#"/>
  </processor>
  <memory>
    <!--
     The hardware `memory` element specifies the available physical memory
     blocks including reserved memory regions (RMRR, see Intel VT-d
     Specification, "8.4 Reserved Memory Region Reporting Structure").

     Only memory blocks reported by the BIOS E820 map as non-*reserved* must
     be configured in this section, e.g. *usable* or *ACPI NVS*, *ACPI data*.
    -->
    <memoryBlock allocatable="false" name="System RAM" physicalAddress="16#0000#" size="16#0009_d000#
     "/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#0010_0000#" size="16#1
     ff0_0000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#2020_0000#" size="16#1
     fe0_4000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#4000_5000#" size="16#6
     ed2_c000#"/>
    <memoryBlock allocatable="false" name="ACPI Non-volatile Storage" physicalAddress="16#bae9_f000#"
      size="16#0010_0000#"/>
    <memoryBlock allocatable="false" name="ACPI Tables" physicalAddress="16#baf9_f000#" size="16#0006
     _0000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#0001_0000_0000#" size="
     16#0003_3e60_0000#"/>
    <reservedMemory name="rmrr1" physicalAddress="16#ba3b_a000#" size="16#0001_7000#">
      <!--
       A `reservedMemory` element is a special memory block declaration. It
       specifies a reserved memory region as outlined in the Intel VT-d
       Specification, "8.4 Reserved Memory Region Reporting Structure" (RMRR).

       Reserved memory regions are BIOS allocated memory ranges that may be DMA
       targets for certain legacy device use-cases. Devices that require access
       to such a region refer to it by name.
      -->
    </reservedMemory>
    <reservedMemory name="rmrr2" physicalAddress="16#bb80_0000#" size="16#0420_0000#"/>
  </memory>
  <devices pciConfigAddress="16#f800_0000#" pciConfigSize="16#0400_0000#">
    <!--
```

```
195       The `devices` element enumerates all devices provided by the hardware
          platform. Different kinds of devices, be it PCI(e) or legacy (non-PCI),
197       can be declared in this section.
        -->
199     <device name="vga">
          <!--
201       The `device` element specifies a physical device and its associated
          resources. There are three main device resource types:
203
          -   IRQ
205
          -   I/O port range
207
          -   Memory
209
          The presence of a PCI element indicates whether the device is a PCI or a
211       legacy device.

213       Capabilities can be used to convey additional device-specific
          information. The base address of the memory mapped PCI config space is
215       defined by the `pciConfigAddress` attribute.
          -->
217       <memory caching="WC" name="buffer" physicalAddress="16#000a_0000#" size="16#0002_0000#">
            <!--
219           A device `memory` element specifies a memory region which is used to
              interact with the associated device.
221
              For PCI devices, the specified region is programmed into one device BAR
223           (Base Address Register) by system firmware. See the PCI Local Bus
              Specification or the PCI Express Base Specification for more details.
225           -->
          </memory>
227       <ioPort end="16#03df#" name="ports" start="16#03c0#">
            <!--
229           The `ioPort` element specifies a device I/O port resource from `start`
              octet up to and including `end` octet. A single byte-accessed port is
231           designated by specifying the same `start` and `end` values.
              -->
233       </ioPort>
        </device>
235     <device name="ps2">
          <irq name="kbd_irq" number="1">
237         <!--
              The `irq` element specifies a device IRQ resource.
239
              The specified IRQ number is one of:
241
              -   Legacy IRQ (ISA)ă
243           Range `0 .. 15`.

245           -   PCI INTx IRQ, line-signaledă
              Range `0 .. Max_LSI_IRQ`, whereas `Max_LSI_IRQ` is defined by the
247           hardware I/O APIC configuration `gsi_base` + `max_redirection_entry`
              of I/O APIC with `max(gsi_base)`. `gsi_base` and
249           `max_redirection_entry` are I/O APIC device capabilities.

251           `msi` sub-elements are present if the device supports MSI interrupts.
              The element count designates the number of supported MSI interrupts.
253           -->
          </irq>
255       <irq name="mouse_irq" number="12"/>
          <ioPort end="16#0060#" name="port_60" start="16#0060#"/>
257       <ioPort end="16#0064#" name="port_64" start="16#0064#"/>
        </device>
259     <device name="cmos_rtc">
          <ioPort end="16#0071#" name="ports" start="16#0070#"/>
261     </device>
        <device name="pcspeaker">
263       <ioPort end="16#0061#" name="Port_61" start="16#0061#"/>
          <ioPort end="16#0043#" name="Port_42_43" start="16#0042#"/>
265     </device>
        <device name="system_board">
267       <!--
            The system board must provide a reset and pm1a_cnt port as well as
269         the pm1a_cnt_slp_typ capability. The presence of this device and
            the necessary resources are checked by the Mucfgvalidate tool.  The
271         resources are used by the kernel for system reboot and poweroff.
          -->
273       <ioPort end="16#0cf9#" name="reset" start="16#0cf9#"/>
          <ioPort end="16#0404#" name="pm1a_cnt" start="16#0404#"/>
275       <capabilities>
            <capability name="systemboard"/>
277         <capability name="pm1a_cnt_slp_typ">7168</capability>
          </capabilities>
279     </device>
        <device name="ioapic_1">
281       <!--
```

```
      The I/O Advanced Programmable Interrupt Controller (I/O APIC) is
      used by the kernel for interrupt routing of legacy IRQs.  The
      presence of this device and the necessary resources are checked by
      the validator tool.
     -->
     <memory caching="UC" name="mem1" physicalAddress="16#fec0_0000#" size="16#1000#"/>
     <capabilities>
      <capability name="ioapic">
       <!--
        A device `capability` is used to assign additional information to a
        device. Such a capability might be used by the Muen toolchain to perform
        certain actions on devices with a given capability (e.g. `ioapic`). A
        system integrator may use this facility to define its own capabilities
        used by custom tools.

        A capability element can have an optional value.
       -->
      </capability>
      <capability name="gsi_base">0</capability>
      <capability name="max_redirection_entry">23</capability>
      <capability name="source_id">16#f0f8#</capability>
     </capabilities>
    </device>
    <device name="iommu_1">
     <!--
      This device specifies an Intel VT-d DMA and interrupt remapping
      hardware. It is used by the Muen SK to implement device separation
      by means of device domains, see below.  The capabilities define
      specific properties of the IOMMU, such as Guest Address Width,
      Fault Register Offset etc. Refer to the Intel VT-d Specification,
      "10.4 Register Descriptions".
     -->
     <memory caching="UC" name="mmio" physicalAddress="16#fed9_0000#" size="16#1000#"/>
     <capabilities>
      <capability name="iommu"/>
      <capability name="agaw">39</capability>
      <capability name="fr_offset">512</capability>
      <capability name="iotlb_invalidate_offset">264</capability>
     </capabilities>
    </device>
    <device name="iommu_2">
     <memory caching="UC" name="mmio" physicalAddress="16#fed9_1000#" size="16#1000#"/>
     <capabilities>
      <capability name="iommu"/>
      <capability name="agaw">39</capability>
      <capability name="fr_offset">512</capability>
      <capability name="iotlb_invalidate_offset">264</capability>
     </capabilities>
    </device>
    <device name="host_bridge_1">
     <description>Intel Corporation 3rd Gen Core processor DRAM Controller</description>
     <pci bus="16#00#" device="16#00#" function="0">
      <!--
       PCI(e) devices are specified using the `pci` element.

       The element provides the following information:

       -   PCI device address (BDF)

       -   Identification

       -   IOMMU group information

       The location of the PCI device in the PCI topology is specified by the
       Bus, Device, Function triplet (BDF).
      -->
      <identification classcode="16#0600#" deviceId="16#0154#" revisionId="16#09#" vendorId="16#8086#
     ">
       <!--
        The `identification` element specifies the PCI device class, device,
        revision and vendor ID.

        For more information, consult the PCI Local Bus Specification,
        "Configuration Space Decoding".
       -->
      </identification>
      <iommuGroup id="0">
       <!--
        Devices in the same IOMMU group cannot be properly isolated from each
        other because they may perform inter-device transactions directly,
        without going through the IOMMU.

        Note that this information is currently not used by the toolchain. It is
        a hint to the system integrator whether two devices can be properly
        isolated from each other or not.
       -->
      </iommuGroup>
```

```
      </pci>
      <memory caching="UC" name="mmconf" physicalAddress="16#f800_0000#" size="16#1000#"/>
     </device>
     <device name="vga_compatible_controller_1">
      <description>Intel Corporation 3rd Gen Core processor Graphics Controller</description>
      <pci bus="16#00#" device="16#02#" function="0">
       <identification classcode="16#0300#" deviceId="16#0166#" revisionId="16#09#" vendorId="16#8086#
      "/>
       <iommuGroup id="1"/>
      </pci>
      <irq name="irq1" number="16">
       <msi name="msi1"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d000_0000#" size="16#0040_0000#"/>
      <memory caching="WC" name="mem2" physicalAddress="16#c000_0000#" size="16#1000_0000#"/>
      <memory caching="WC" name="mem3" physicalAddress="16#000c_0000#" size="16#0002_0000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f801_0000#" size="16#1000#"/>
      <ioPort end="16#603f#" name="ioport1" start="16#6000#"/>
      <reservedMemory ref="rmrr2">
       <!--
        This device specifies that it requires access to the reserved
        memory range (RMRR) with the given name.
       -->
      </reservedMemory>
     </device>
     <device name="usb_controller_1">
      <description>Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Host Controller</
      description>
      <pci bus="16#00#" device="16#14#" function="0">
       <identification classcode="16#0c03#" deviceId="16#1e31#" revisionId="16#04#" vendorId="16#8086#
      "/>
       <iommuGroup id="2"/>
      </pci>
      <irq name="irq1" number="16">
       <msi name="msi1">
        <!--
         There are two different interrupt types which devices may trigger:
         legacy/PCI LSI IRQs and Message Signaled Interrupts (MSI). The
         legacy/PCI LSI IRQ is specified by the number attribute of the `irq`
         element. For MSIs, each `msi` element defines an MSI IRQ that may be
         assigned to subjects. Each MSI may be individually routed.
        -->
       </msi>
       <msi name="msi2"/>
       <msi name="msi3"/>
       <msi name="msi4"/>
       <msi name="msi5"/>
       <msi name="msi6"/>
       <msi name="msi7"/>
       <msi name="msi8"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d252_0000#" size="16#0001_0000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f80a_0000#" size="16#1000#"/>
      <reservedMemory ref="rmrr1"/>
     </device>
     <device name="communication_controller_1">
      <description>Intel Corporation 7 Series/C216 Chipset Family MEI Controller #1</description>
      <pci bus="16#00#" device="16#16#" function="0">
       <identification classcode="16#0780#" deviceId="16#1e3a#" revisionId="16#04#" vendorId="16#8086#
      "/>
       <iommuGroup id="3"/>
      </pci>
      <irq name="irq1" number="16">
       <msi name="msi1"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d253_5000#" size="16#1000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f80b_0000#" size="16#1000#"/>
     </device>
     <device name="serial_controller_1">
      <description>Intel Corporation 7 Series/C210 Series Chipset Family KT Controller</description>
      <pci bus="16#00#" device="16#16#" function="3">
       <identification classcode="16#0700#" deviceId="16#1e3d#" revisionId="16#04#" vendorId="16#8086#
      "/>
       <iommuGroup id="3"/>
      </pci>
      <irq name="irq1" number="19">
       <msi name="msi1"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d253_c000#" size="16#1000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f80b_3000#" size="16#1000#"/>
      <ioPort end="16#60b7#" name="ioport1" start="16#60b0#"/>
     </device>
     <device name="ethernet_controller_1">
      <description>Intel Corporation 82579LM Gigabit Network Connection (Lewisville)</description>
      <pci bus="16#00#" device="16#19#" function="0">
       <identification classcode="16#0200#" deviceId="16#1502#" revisionId="16#04#" vendorId="16#8086#
      "/>
```

```
      <iommuGroup id="4"/>
     </pci>
     <irq name="irq1" number="20">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d250_0000#" size="16#0002_0000#"/>
     <memory caching="UC" name="mem2" physicalAddress="16#d253_b000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80c_8000#" size="16#1000#"/>
     <ioPort end="16#609f#" name="ioport1" start="16#6080#"/>
    </device>
    <device name="usb_controller_2">
     <description>Intel Corporation 7 Series/C216 Chipset Family USB Enhanced Host Controller #2</
     description>
     <pci bus="16#00#" device="16#1a#" function="0">
      <identification classcode="16#0c03#" deviceId="16#1e2d#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="5"/>
     </pci>
     <irq name="irq1" number="16"/>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_a000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80d_0000#" size="16#1000#"/>
     <reservedMemory ref="rmrr1"/>
    </device>
    <device name="audio_device_1">
     <description>Intel Corporation 7 Series/C216 Chipset Family High Definition Audio Controller</
     description>
     <pci bus="16#00#" device="16#1b#" function="0">
      <identification classcode="16#0403#" deviceId="16#1e20#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="6"/>
     </pci>
     <irq name="irq1" number="22">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_0000#" size="16#4000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80d_8000#" size="16#1000#"/>
    </device>
    <device name="usb_controller_3">
     <description>Intel Corporation 7 Series/C216 Chipset Family USB Enhanced Host Controller #1</
     description>
     <pci bus="16#00#" device="16#1d#" function="0">
      <identification classcode="16#0c03#" deviceId="16#1e26#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="11"/>
     </pci>
     <irq name="irq1" number="23"/>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_9000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80e_8000#" size="16#1000#"/>
     <reservedMemory ref="rmrr1"/>
    </device>
    <device name="isa_bridge_1">
     <description>Intel Corporation QM77 Express Chipset LPC Controller</description>
     <pci bus="16#00#" device="16#1f#" function="0">
      <identification classcode="16#0601#" deviceId="16#1e55#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="12"/>
     </pci>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80f_8000#" size="16#1000#"/>
    </device>
    <device name="sata_controller_1">
     <description>Intel Corporation 7 Series Chipset Family 6-port SATA Controller [AHCI mode]</
     description>
     <pci bus="16#00#" device="16#1f#" function="2">
      <identification classcode="16#0106#" deviceId="16#1e03#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="12"/>
     </pci>
     <irq name="irq1" number="19">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_8000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80f_a000#" size="16#1000#"/>
     <ioPort end="16#60af#" name="ioport1" start="16#60a8#"/>
     <ioPort end="16#60bf#" name="ioport2" start="16#60bc#"/>
     <ioPort end="16#60a7#" name="ioport3" start="16#60a0#"/>
     <ioPort end="16#60bb#" name="ioport4" start="16#60b8#"/>
     <ioPort end="16#607f#" name="ioport5" start="16#6060#"/>
    </device>
    <device name="smbus_1">
     <description>Intel Corporation 7 Series/C216 Chipset Family SMBus Controller</description>
     <pci bus="16#00#" device="16#1f#" function="3">
      <identification classcode="16#0c05#" deviceId="16#1e22#" revisionId="16#04#" vendorId="16#8086#
     "/>
      <iommuGroup id="12"/>
     </pci>
     <irq name="irq1" number="18"/>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_4000#" size="16#1000#"/>
```

```
      <memory caching="UC" name="mmconf" physicalAddress="16#f80f_b000#" size="16#1000#"/>
      <ioPort end="16#efbf#" name="ioport1" start="16#efa0#"/>
     </device>
     <device name="network_controller_1">
      <description>Intel Corporation Centrino Advanced-N 6205 [Taylor Peak]</description>
      <pci bus="16#03#" device="16#00#" function="0">
       <identification classcode="16#0280#" deviceId="16#0085#" revisionId="16#34#" vendorId="16#8086#
      "/>
       <iommuGroup id="13"/>
      </pci>
      <irq name="irq1" number="17">
       <msi name="msi1"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d1c0_0000#" size="16#2000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f830_0000#" size="16#1000#"/>
     </device>
     <device name="system_peripheral_1">
      <description>Ricoh Co Ltd PCIe SDXC/MMC Host Controller</description>
      <pci bus="16#04#" device="16#00#" function="0">
       <identification classcode="16#0880#" deviceId="16#e823#" revisionId="16#07#" vendorId="16#1180#
      "/>
       <iommuGroup id="14"/>
      </pci>
      <irq name="irq1" number="18">
       <msi name="msi1"/>
      </irq>
      <memory caching="UC" name="mem1" physicalAddress="16#d140_0000#" size="16#1000#"/>
      <memory caching="UC" name="mmconf" physicalAddress="16#f840_0000#" size="16#1000#"/>
     </device>
    </devices>
   </hardware>
   <platform>
    <!--
     To enable a uniform view of the hardware resources across different
     physical machines from the system integrators perspective, the platform
     description layer is interposed between the hardware resource
     description and the rest of the system policy. This allows to build a
     Muen system for different physical target machines using the same system
     policy.
    -->
    <mappings>
     <!--
      Platform device alias and class mappings section. Used to assign a
      stable name to a hardware device or to group (multiple) devices under a
      given name.
     -->
     <aliases>
      <!--
       Aliases are a renaming mechanism for physical hardware devices and their
       resources. By using alias names in the system policy references to
       concrete hardware resources can be avoided. Additionally, aliases may be
       used to define a device which only contains a subset of the resources of
       the physical device. This can be achieved by only renaming the resources
       that the device alias should export.
      -->
      <alias name="serial_device_1" physical="serial_controller_1">
       <resource name="ioport1" physical="ioport1"/>
      </alias>
      <alias name="nic_1" physical="ethernet_controller_1">
       <resource name="irq1" physical="irq1">
        <resource name="msi1" physical="msi1"/>
       </resource>
       <resource name="mem1" physical="mem1"/>
       <resource name="mem2" physical="mem2"/>
      </alias>
      <alias name="storage_controller" physical="sata_controller_1"/>
      <alias name="ahci_controller" physical="sata_controller_1">
       <resource name="irq1" physical="irq1">
        <resource name="msi1" physical="msi1"/>
       </resource>
       <resource name="ahci_registers" physical="mem1"/>
       <resource name="mmconf" physical="mmconf"/>
      </alias>
     </aliases>
     <classes>
      <!--
       The `classes` element specifies a list of device classes.
      -->
      <class name="desktop_devices">
       <!--
        Device classes enable the grouping of devices and allow referencing all
        devices by a single name. This simplifies the process of assigning
        multiple devices to a subject.

        Note: A device class may contain an arbitrary number of devices,
        including zero.
       -->
```

```
      <device physical="audio_device_1"/>
      <device physical="ethernet_controller_1"/>
      <device physical="network_controller_1"/>
      <device physical="sata_controller_1"/>
      <device physical="system_peripheral_1"/>
     </class>
     <class name="additional_nics">
      <device physical="network_controller_1"/>
     </class>
    </classes>
   </mappings>
   <kernelDiagnostics type="uart">
    <!--
     The debug build Muen SK can be instructed to output debugging
     information during runtime. The platform diagnostics device specifies
     which device the kernel is to use for this purpose.

     The presence of this device and the necessary resources are checked by
     the validator tool.
    -->
    <device physical="serial_controller_1">
     <ioPort physical="ioport1"/>
    </device>
   </kernelDiagnostics>
  </platform>
  <memory>
   <!--
    This section declares all physical memory regions (RAM) and thus the
    physical memory layout of the system. Regions declared in this section
    can be assigned to subjects and device domains.

    Memory regions are defined by the following attributes:

    -   Name

    -   Caching type

    -   Size

    -   Physical address\*

    -   Alignment\*

    -   Memory type\*

    Attributes with an asterisk are optional. While alignment and memory
    type are set to a default value if not specified, the physical address
    is filled in by the allocator tool, which allocates all memory regions
    and finalizes the physical memory layout.

    Additionally, the content of a region can be declared as backed by a
    file or filled with a pattern.

    Note: The caching type is an attribute of the physical memory region by
    design to avoid inconsistent typing, even though the Intel Page
    Attribute Table (PAT) mechanism allows to set it for each memory
    mapping, see Intel SDM Vol. 3A, "11.12.4 Programming the PAT".
   -->
   <memory caching="WB" name="control_example" size="16#1000#">
    <fill pattern="16#00#">
     <!--
      The 'fill' element designates a memory region which is initialized with
      the given pattern.
     -->
    </fill>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_sm_1" size="16#1000#">
    <fill pattern="16#00#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_sm_2" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_time" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_linux_1" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_example" size="16#1000#">
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_sm_1" size="16#1000#">
```

```
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_sm_2" size="16#1000#">
    <fill pattern="16#00#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_time" size="16#1000#">
    <fill pattern="16#00#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_linux_1" size="16#1000#">
    <fill pattern="16#00#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="initramfs" size="16#0113_0000#" type="subject_initrd">
    <file filename="initramfs.cpio.gz" offset="none">
     <!--
      The 'file' child element designates a file-backed memory region.

      The 'filename' attribute specifies the name of the file to use as
      content for the physical memory region, the 'offset' attribute is 'none'
      by default but can be customized to include a partial file.
     -->
    </file>
   </memory>
   <memory caching="WB" name="nic_linux|ram" size="16#1000_0000#"/>
   <memory caching="WB" name="nic_linux|lowmem" size="16#0008_0000#"/>
   <memory caching="WB" name="storage_linux|ram" size="16#1000_0000#"/>
   <memory caching="WB" name="storage_linux|lowmem" size="16#0008_0000#"/>
   <memory caching="WB" name="example_filled_region" size="16#1000#">
    <fill pattern="16#5a#"/>
   </memory>
   <memory caching="UC" name="crash_audit" physicalAddress="16#0001_00a1_1000#" size="16#1000#" type=
     "subject_crash_audit">
   </memory>
  </memory>
  <deviceDomains>
   <!--
    The physical memory accessible by PCI devices is specified by so called
    device domains. Such domains define memory mappings of physical memory
    regions for one or multiple devices. Device references select a subset
    of hardware devices provided by the hardware/platform. Devices may be
    referenced by device name, alias or device class.

    Device references can optionally set the 'mapReservedMemory' attribute
    so RMRR regions referenced by the device are also mapped into the device
    domain.

    Device domains are isolated from each other by the use of Intel VT-d.
   -->
   <domain name="nic_domain">
    <memory>
     <mapSubjectMemory subject="nic_linux"/>
    </memory>
    <devices>
     <device logical="first_nic" physical="ethernet_controller_1"/>
     <device logical="additional_nics" physical="additional_nics"/>
    </devices>
   </domain>
   <domain name="storage_domain">
    <memory>
     <memory executable="false" logical="dma1" physical="storage_linux|lowmem" virtualAddress="
     16#0002_0000#" writable="true">
      <!--
       A 'memory' element maps a physical memory region into the address space
       of a device domain or subject entity. The region will be accessible to
       the entity at the specified 'virtualAdress' with permissions defined by
       the 'executable' and 'writable' attributes.
      -->
     </memory>
     <memory executable="false" logical="dma2" physical="storage_linux|ram" virtualAddress="16#0100
     _0000#" writable="true"/>
    </memory>
    <devices>
     <device logical="storage_controller" physical="storage_controller"/>
     <device logical="xhci" physical="usb_controller_1"/>
    </devices>
   </domain>
  </deviceDomains>
  <events>
   <!--
    Events are an activity caused by a subject (source) that impacts a
    second subject (target) or is directed at the kernel. Events are
    declared globally and have a unique name to be unambiguous. An event
    must have a single source and one target.

    Subjects can use events to either deliver an interrupt, hand over
```

```
    execution to or reset the state of a target subject. The first kind of
783 event provides a basic notification mechanism and enables the
    implementation of event-driven services. The second type facilitates
785 suspension of execution of the source subject and switching to the
    target. Such a construct is used to pass the thread of execution on to a
787 different subject, e.g. invocation of a debugger subject if an error
    occurs in the source subject. The third kind is used to facilitate the
789 restart of subjects.

791 An event can also have the same source and target, which is called
    *self* event. Such events are useful to implement para-virtualized
793 timers in VM subjects for example.

795 Kernel events are special in that they are targeted at the kernel. The
    currently supported events are system reboot and shutdown.
797 -->
   <event mode="switch" name="resume_linux_1">
799  <!--
     The `eventType` specifies an event by name and mode.
801

     The following event modes are currently supported:
803

     -   `asap`ă
805  The asap event is an abstraction to state that the event should be
     delivered as soon as possible, depending on the CPU of the target
807  subject. If the target runs on another CPU core, this mode is
     expanded to mode *ipi*, which is only available in policy formats A
809  and B, instructing the kernel to preempt the kernel running the
     target subject and inject the event immediately. If the target
811  subject runs on the same core as the source subject, the mode is
     expanded to mode *async*.
813

     -   `async`ă
815  Async events trigger no preemption at the target subject. The event
     is marked as pending in the target subjects pending event table and
817  inserted on the next VM exit/entry cycle of the target subject.

819  -   `self`ă
     An event can also have the same source and target, which is called a
821  self event. Such events are useful to implement para-virtualized
     timers in VM subjects for example. A subject sends itself a delayed
823  event, using the timed event mechanism. Note that a self event must
     always have a target action assigned, which is checked by the
825  validator.

827  -   `switch`ă
     The switch mode facilitates suspension of execution of the source
829  subject and switching to the target. This can only happen between
     subjects running on the same core. Such a construct is used to pass
831  the thread of execution on to a different subject, e.g. invocation
     of a debugger subject if an error occurs in the source subject. It
833  is called *handover* or *handover event*.

835  -   `kernel`ă
     These kinds of events are directed at the kernel an thus only
837  specify a source since the target is the kernel. They are used to
     enable specific subjects to unmask level-triggered IRQs and trigger
839  a system reboot, poweroff or explicit panic (crash audit slot
     allocation and reboot).
841  -->
   </event>
843 <event mode="switch" name="resume_linux_2"/>
   <event mode="switch" name="trap_to_sm_1"/>
845 <event mode="switch" name="trap_to_sm_2"/>
   <event mode="switch" name="load_linux_1"/>
847 <event mode="switch" name="start_linux_1"/>
   <event mode="switch" name="reset_linux_1"/>
849 <event mode="switch" name="reset_linux_2"/>
   <event mode="async" name="reset_sm_1"/>
851 <event mode="async" name="reset_slot_1"/>
   <event mode="async" name="request_reset_slot_1"/>
853 <event mode="async" name="serial_irq4_linux_1"/>
   <event mode="async" name="serial_irq4_linux_2"/>
855 <event mode="self" name="timer_linux_1"/>
   <event mode="self" name="timer_linux_2"/>
857 <event mode="kernel" name="subject_sleep"/>
   <event mode="kernel" name="subject_yield"/>
859 <event mode="kernel" name="system_reboot"/>
   <event mode="kernel" name="system_poweroff"/>
861 <event mode="kernel" name="system_panic"/>
   <event mode="self" name="example_self"/>
863 </events>
   <channels>
865  <!--
     Inter-subject communication is specified by so called channels. These
867  channels represent directed information flows since they have a single
     writer and possibly multiple readers. Optionally a channel can have an
```

```
    associated notification event (doorbell interrupt).

    Channels are declared globally and have an unique name to be
    unambiguous.

    Note that channels are a policy source format abstraction. The toolchain
    resolves this concept into memory regions and events as well as the
    appropriate subject mappings.
   -->
   <channel hasEvent="asap" name="input_events" size="16#1000#">
    <!--
     The `channel` element declares a physical channel.

     Besides the `name` and `size` of the channel, the optional `hasEvent`
     attribute can be set to declare that the given channel requests an
     associated event. The expander tool will then automatically create a
     global event of the requested event type.
    -->
   </channel>
   <channel hasEvent="asap" name="virtual_input_1" size="16#1000#"/>
   <channel hasEvent="asap" name="virtual_input_2" size="16#1000#"/>
   <channel hasEvent="asap" name="virtual_console_1" size="16#0001_0000#"/>
   <channel hasEvent="asap" name="virtual_console_2" size="16#0001_0000#"/>
   <channel name="time_info" size="16#1000#"/>
   <channel name="debuglog_subject1" size="16#0002_0000#"/>
   <channel name="debuglog_subject2" size="16#0002_0000#"/>
   <channel name="debuglog_subject3" size="16#0002_0000#"/>
   <channel name="debuglog_subject4" size="16#0002_0000#"/>
   <channel name="debuglog_subject5" size="16#0002_0000#"/>
   <channel name="debuglog_subject6" size="16#0002_0000#"/>
   <channel name="debuglog_subject7" size="16#0002_0000#"/>
   <channel hasEvent="switch" name="nic_dm_request" size="16#1000#"/>
   <channel hasEvent="switch" name="nic_dm_response" size="16#1000#"/>
   <channel hasEvent="switch" name="storage_dm_request" size="16#1000#"/>
   <channel hasEvent="switch" name="storage_dm_response" size="16#1000#"/>
   <channel name="debuglog_controller" size="16#0002_0000#"/>
   <channel name="testchannel_1" size="16#1000#"/>
   <channel name="testchannel_2" size="16#1000#"/>
   <channel name="testchannel_3" size="16#0010_0000#"/>
   <channel name="testchannel_4" size="16#0010_0000#"/>
   <channel name="debuglog_example" size="16#0002_0000#"/>
   <channel hasEvent="asap" name="example_request" size="16#1000#"/>
   <channel hasEvent="asap" name="example_response" size="16#1000#"/>
  </channels>
  <components>
   <!--
    The `components` element holds a list of components and component
    libraries.

    Note that components are a policy source format abstraction. The
    toolchain resolves this concept into subjects by adding the appropriate
    memory regions, events and devices.
   -->
   <library name="libmucontrol">
    <!--
     A component library is a specialized component specification which is
     used to share common resources required for library code to operate.
     Component libraries can be included by multiple components in order to
     share functionality. An example is a logging service provided by a
     dedicated component, whereas the logging client is provided as a library
     with a shared memory channel for the actual log messages.

     A component specification declares library dependencies to request the
     library resources from the system through the inclusion of the library
     specification in the `depends` section. This way components inherit the
     resources of libraries.

     On the source code level, a library is included by mechanisms provided
     by the respective programming language. Note that the component library
     code is *not* shared between components but lives in the isolated
     execution environment of a subject instantiating the component (i.e.
     statically linked libraries).

     Libraries can request the same resources as ordinary components. A
     subject instantiating the component must also map the resources
     requested by libraries the component depends on.
    -->
    <requires>
     <memory>
      <!--
       In this section, components can specify expected memory mappings with
       given access rights and region size.
      -->
      <memory executable="false" logical="control" size="16#1000#" virtualAddress="16#000f_ffff_3000#
     " writable="false">
       <!--
        The `memory` element requests a memory region with the specified `size`
```

```
955         and permissions from the system. The region is expected to be placed at
            the address given via the `virtualAddress` attribute.
957         -->
          </memory>
959       <memory executable="false" logical="status" size="16#1000#" virtualAddress="16#000f_ffff_2000#"
          writable="true"/>
        </memory>
961     </requires>
      </library>
963   <library name="libmudebuglog">
       <config>
965     <string name="logchannel_size" value="16#0002_0000#"/>
       </config>
967    <requires>
        <channels>
969      <!--
          Components and libraries use the `channels` sub-section of `requires` to
971       specify expected communication channels.
          -->
973       <writer logical="debuglog" size="16#0002_0000#" virtualAddress="16#000f_fff0_0000#">
           <!--
975

           -->
977       </writer>
        </channels>
979    </requires>
      </library>
981   <library name="libmudm">
       <requires>
983     <channels>
         <writer event="8" logical="dm_pciconf_req" size="16#1000#" virtualAddress="16#2000_0000#"/>
985      <reader logical="dm_pciconf_res" size="16#1000#" virtualAddress="16#2000_1000#">
          <!--
987        The `reader` element requests a read-only channel of the specified size,
           address and optional notification vector.
989        -->
         </reader>
991     </channels>
       </requires>
993   </library>
      <library name="libmuinit">
995    <depends>
        <library ref="libmucontrol"/>
997    </depends>
      </library>
999   <library name="libmutime">
       <requires>
1001     <channels>
          <reader logical="time_info" size="16#1000#" virtualAddress="16#000f_ffd0_0000#"/>
1003     </channels>
       </requires>
1005  </library>
      <library name="libxhcidbg">
1007   <requires>
        <memory>
1009     <memory executable="false" logical="xhci_dma" size="16#0004_1000#" virtualAddress="16#0100_0000
        #" writable="true"/>
        </memory>
1011    <devices>
         <device logical="xhci">
1013      <memory executable="false" logical="xhci_registers" size="16#0001_0000#" virtualAddress="16#
        e000_0000#" writable="true"/>
         </device>
1015    </devices>
       </requires>
1017  </library>
      <library name="muinit">
1019   <depends>
        <library ref="libmuinit"/>
1021   </depends>
       <requires>
1023    <vcpu>
         <!--
1025      The `vcpu` element controls the execution behavior of the virtual CPU
          (vCPU). A default vCPU profile is selected by the component profile, but
1027      CPU execution settings can be customized both at component and subject
          level.
1029      -->
         <registers>
1031      <gpr>
           <rip>16#0010_0000#</rip>
1033      </gpr>
         </registers>
1035    </vcpu>
       </requires>
1037   <provides>
```

```
      <memory executable="true" logical="muinit" size="16#9000#" type="subject_binary" virtualAddress=
      "16#0010_0000#" writable="false">
       <file filename="muinit" offset="none"/>
      </memory>
     </provides>
    </library>
    <component name="ahci_drv" profile="native">
     <!--
      A component is a piece of software which shall be executed by the SK.
      Components represent the building blocks of a component-based system and
      can be regarded as templates for executable entities instantiated by
      subjects.

      The specification of a component declares the *binary program* by means
      of (file-backed memory) regions. It also specifies the components view
      of the expected execution environment. A component may request the
      following resources from the system:

      -   Logical channels

      -   Logical memory regions

      -   Logical devices

      -   Logical events

      Components are identified by name and specify a profile. The profile
      controls the settings of the virtual CPU (vCPU).
     -->
     <depends>
      <!--
       Components and libraries are allowed to declare dependencies to other
       libraries. All resources required by the included library are merged
       with the ones specified by the component or library. Libraries can
       depend on other libraries.

       A subject realizing this component must correctly map all component and
       library resource requirements to physical resources in order to fulfill
       the expectations.
      -->
      <library ref="libmudebuglog"/>
     </depends>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <memory>
       <!-- for 32 ports 16#c000# bytes are needed for descriptor tables + 16K for device init -->
       <memory executable="false" logical="dma_region" size="16#0001_0000#" virtualAddress="16#
      a000_0000#" writable="true"/>
       <array elementSize="16#0100_0000#" executable="false" logical="blockdev_shm" virtualAddressBase
      ="16#a100_0000#" writable="true">
        <memory logical="blockdev_shm1"/>
        <memory logical="blockdev_shm2"/>
       </array>
      </memory>
      <channels>
       <array elementSize="16#0000_8000#" logical="blockdev_request" vectorBase="64"
      virtualAddressBase="16#0001_0000_0000#">
        <!--
         The channel array abstraction simplifies the declaration of consecutive
         channel mappings with a given base address, channel size and optional
         event/vector bases. The child elements declare the number of expected
         channels and either the `reader` or `writer` role.
        -->
        <reader logical="blockdev_request1">
         <!--
          Array entries specify the number of array elements and assign a logical
          name to each element.
         -->
        </reader>
        <reader logical="blockdev_request2"/>
       </array>
       <array elementSize="16#0000_4000#" eventBase="16" logical="blockdev_response"
      virtualAddressBase="16#0001_0001_0000#">
        <writer logical="blockdev_response1"/>
        <writer logical="blockdev_response2"/>
       </array>
      </channels>
      <devices>
       <!--
        The `devices` sub-section of the `requires` section is used to specify
        expected devices with their associated resources.
```

```
      -->
      <device logical="ahci_controller">
       <!--
        A 'device' element specifies an expected logical device with its
        resources. Possible resources are 'irq', 'memory' and 'ioPort'.
       -->
       <irq logical="irq" vector="48">
        <!--
         An 'irq' element of a logical device reference requests an IRQ with
         given number from the system policy. The specified number will be
         injected when the device requires attention for the associated logical
         function.
        -->
       </irq>
       <memory executable="false" logical="ahci_registers" size="16#1000#" virtualAddress="16#
      e000_0000#" writable="true"/>
       <memory executable="false" logical="mmconf" size="16#1000#" virtualAddress="16#f800_8000#"
      writable="true"/>
      </device>
     </devices>
    </requires>
    <provides>
     <!--
      Components usually come in the form of an executable file. To this end,
      the 'provides' section specifies the memory regions of the component
      binary executable with their content.

      From a security perspective, it is often desirable to provide the
      different binary section as separate memory regions with the appropriate
      access rights, i.e. only the text section is executable, rodata is not
      writable and so on.

      Memory specified in this sections are expanded to mapped physical
      regions for each subject that instantiates this component.

      Note: the Mucbinsplit tool can be used to extract these section from an
      ELF binary into separate files and automatically add the corresponding
      memory elements to the component specification.
     -->
     <memory executable="true" logical="text" size="16#9000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
      <!--
       A 'memory' element in the 'provides' section declares memory region
       provided by the component. Mostly used to provide (a part) of the
       component binary.
      -->
      <file filename="ahci_drv_text" offset="none"/>
      <hash value="16#270e3253624032bafa782340acf26b056ca2e635ecfdaaa28bf6c74af3cfea60#"/>
     </memory>
     <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary" virtualAddress
     ="16#0020_9000#" writable="false">
      <file filename="ahci_drv_rodata" offset="none"/>
      <hash value="16#f7eba385de0bba39cc5d5e860fca26c765ed0fd05b8ba326639e5c699341251a#"/>
     </memory>
     <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_a000#" writable="true">
      <file filename="ahci_drv_data" offset="none"/>
      <hash value="16#1cb2148aef42e097dad59630d8940dce44f8b351f88fcfac9ded03d0a2a831b1#"/>
     </memory>
     <memory executable="false" logical="bss" size="16#3000#" type="subject_binary" virtualAddress="
     16#0020_b000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
     <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress=
     "16#1000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
    </provides>
   </component>
   <component name="controller" profile="native">
    <depends>
     <library ref="libmudebuglog"/>
    </depends>
    <requires>
     <vcpu>
      <registers>
       <gpr>
        <rip>16#0020_0000#</rip>
       </gpr>
      </registers>
     </vcpu>
     <memory>
      <array elementSize="16#1000#" executable="false" logical="control" virtualAddressBase="16#0001
     _0000_0000#" writable="true">
       <memory logical="control_1"/>
       <memory logical="control_2"/>
       <memory logical="control_3"/>
```

```
      <memory logical="control_4"/>
      <memory logical="control_5"/>
     </array>
     <array elementSize="16#1000#" executable="false" logical="status" virtualAddressBase="16#0001
    _0000_5000#" writable="false">
      <memory logical="status_1"/>
      <memory logical="status_2"/>
      <memory logical="status_3"/>
      <memory logical="status_4"/>
      <memory logical="status_5"/>
     </array>
    </memory>
    <events>
     <!--
      The `events` sub-section of the `requires` section is used to specify
      expected events with optional event actions.

      A component can specify both source as well as target events.
     -->
     <source>
      <event id="10" logical="reset_slot_1_sm"/>
      <event id="11" logical="reset_slot_1_linux"/>
     </source>
     <target>
      <event logical="request_reset_slot_1">
       <inject_interrupt vector="32"/>
      </event>
     </target>
    </events>
   </requires>
   <provides>
    <memory executable="true" logical="text" size="16#2000#" type="subject_binary" virtualAddress="
    16#0020_0000#" writable="false">
     <file filename="controller_text" offset="none"/>
     <hash value="16#5f81491568d860bd9b42f6c6d2b49733eabf65069ba18b79031d7bcb95ef0f49#"/>
    </memory>
    <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary" virtualAddress
    ="16#0020_2000#" writable="false">
     <file filename="controller_rodata" offset="none"/>
     <hash value="16#df41e836ab51453074f093d2824347d90d272ec8abaf4197c5ddb7636fd83320#"/>
    </memory>
    <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress="
    16#0020_3000#" writable="true">
     <file filename="controller_data" offset="none"/>
     <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
    </memory>
    <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress="
    16#0020_4000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
    <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress=
    "16#1000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
   </provides>
  </component>
  <component name="dbgserver" profile="native">
   <config>
    <!--
     Components may declare their own configuration values in the config
     section. Just like global system config values, these can also be
     used in `<if>` expressions and XML attribute value expansion.
    -->
     <boolean name="sink_serial" value="true"/>
     <boolean name="sink_shmem" value="false"/>
     <boolean name="hsuart_enabled" value="false"/>
     <boolean name="sink_xhcidbg" value="false"/>
     <boolean name="default_channel_enabled_state" value="true"/>
     <boolean name="sink_pcspkr" value="false"/>
     <string name="logchannel_size" value="16#0002_0000#"/>
     <string name="debugconsole_port_start" value="16#60b0#"/>
     <string name="enabled_channels_override" value=""/>
     <string name="debugconsole_port_end" value="16#60b7#"/>
    </config>
    <requires>
     <vcpu>
      <registers>
       <gpr>
        <rip>16#0020_0000#</rip>
       </gpr>
      </registers>
     </vcpu>
     <memory>
      <memory executable="false" logical="crash_audit" size="16#1000#" virtualAddress="16#0001
    _0000_0000#" writable="false"/>
     </memory>
     <channels>
```

```
      <array elementSize="16#0002_0000#" logical="log_channels" virtualAddressBase="16#a000_0000#">
       <reader logical="log_channel1"/>
       <reader logical="log_channel2"/>
       <reader logical="log_channel3"/>
       <reader logical="log_channel4"/>
       <reader logical="log_channel5"/>
       <reader logical="log_channel_example"/>
       <reader logical="log_channel_6"/>
       <reader logical="log_channel7"/>
       <reader logical="log_channel8"/>
      </array>
     </channels>
     <devices>
      <device logical="debugconsole">
       <ioPort end="16#60b7#" logical="port" start="16#60b0#">
        <!--
         The `ioPort` element requests a device I/O port resource with given
         range `start .. end` from the system.
        -->
       </ioPort>
      </device>
     </devices>
     <events>
      <source>
       <event id="30" logical="shutdown">
        <system_poweroff>
         <!--
          An example of a source event action directed at the kernel. If this
          event is triggered by the associated subject, the system will power
          off.
         -->
        </system_poweroff>
       </event>
       <event id="31" logical="reboot">
        <system_reboot/>
       </event>
      </source>
     </events>
    </requires>
    <provides>
     <memory executable="true" logical="text" size="16#a000#" type="subject_binary" virtualAddress="
    16#0020_0000#" writable="false">
      <file filename="dbgserver_text" offset="none"/>
      <hash value="16#c8da027b04d470c6e960e4e4178007dec2717d5150206c5feeb5876239d1dce6#"/>
     </memory>
     <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
    virtualAddress="16#0020_a000#" writable="false">
      <file filename="dbgserver_rodata" offset="none"/>
      <hash value="16#e02e60e1a1f04bf00c7cd5f6e95fd1ff443e55425f5b3e4e5f9bc3b19e7b0aff#"/>
     </memory>
     <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
    "16#0020_b000#" writable="true">
      <file filename="dbgserver_data" offset="none"/>
      <hash value="16#d43fbd74e74608f7b7a3279e6a4c24c710752ac5f074a490e5a47705698a1bc1#"/>
     </memory>
     <memory executable="false" logical="bss" size="16#e000#" type="subject_binary" virtualAddress="
    16#0020_c000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
     <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
    ="16#1000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
    </provides>
   </component>
   <component name="dm" profile="native">
    <depends>
     <library ref="libmudebuglog"/>
    </depends>
    <requires>
     <vcpu>
      <registers>
       <gpr>
        <rip>16#0020_0000#</rip>
       </gpr>
      </registers>
     </vcpu>
     <channels>
      <reader logical="request" size="16#1000#" virtualAddress="16#0001_0000_0000#"/>
      <writer event="16" logical="response" size="16#1000#" virtualAddress="16#0001_0000_1000#"/>
     </channels>
    </requires>
    <provides>
     <memory executable="true" logical="text" size="16#4000#" type="subject_binary" virtualAddress="
    16#0020_0000#" writable="false">
      <file filename="dm_text" offset="none"/>
      <hash value="16#dc81a95b073a39da9a537ed6c67267eeaff6e054540ee5aed42651fd0b6ea4cd#"/>
```

```
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
     virtualAddress="16#0020_4000#" writable="false">
        <file filename="dm_rodata" offset="none"/>
        <hash value="16#9b0f0a96844d7684153e7b7ff87212861f47de003b5f73881f4f1150c27fe686#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
     "16#0020_5000#" writable="true">
        <file filename="dm_data" offset="none"/>
        <hash value="16#9348d0aec58fa2e80f74bf3a52440ec842aa7e6bf948bcd98d40d1c30dc218ec#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_6000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
    </provides>
  </component>
  <component name="example" profile="native">
    <config>
      <boolean name="ahci_drv_enabled" value="false"/>
      <boolean name="print_serial" value="false"/>
      <boolean name="print_vcpu_speed" value="true"/>
      <integer name="serial" value="123456789"/>
      <string name="greeter" value="Subject running"/>
    </config>
    <depends>
      <library ref="libmudebuglog"/>
      <library ref="muinit"/>
    </depends>
    <requires>
      <vcpu>
        <vmx>
          <masks>
            <exception>
              <Breakpoint>0</Breakpoint>
            </exception>
          </masks>
        </vmx>
        <registers>
          <gpr>
            <rip>16#0020_0000#</rip>
          </gpr>
          <cr4>
            <XSAVEEnable>1</XSAVEEnable>
          </cr4>
        </registers>
      </vcpu>
      <memory>
        <memory executable="false" logical="filled_region" size="16#1000#" virtualAddress="16#0001
     _0000_0000#" writable="true"/>
      </memory>
      <channels>
        <reader logical="example_request" size="16#1000#" vector="64" virtualAddress="16#0001
     _0000_1000#"/>
        <writer event="16" logical="example_response" size="16#1000#" virtualAddress="16#0001
     _0000_2000#"/>
      </channels>
      <events>
        <source>
          <event id="2" logical="yield">
            <subject_yield/>
          </event>
          <event id="3" logical="timer"/>
          <event id="4" logical="sleep">
            <subject_sleep/>
          </event>
        </source>
        <target>
          <event logical="inject_timer">
            <inject_interrupt vector="37"/>
          </event>
        </target>
      </events>
    </requires>
    <provides>
      <memory executable="false" logical="interrupt_stack" size="16#2000#" type="subject_binary"
     virtualAddress="16#0001_0000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
      <memory executable="true" logical="text" size="16#4000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
        <file filename="example_text" offset="none"/>
        <hash value="16#f09a98fdd53015ba2c2484b330b68bbad129d60054a6f610f26e9efe300fb379#"/>
```

```
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
     virtualAddress="16#0020_4000#" writable="false">
       <file filename="example_rodata" offset="none"/>
       <hash value="16#c647749cba2a151be2a1c451441bfc4882e76f8a554c74ee497dfdcc55b70785#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
     "16#0020_5000#" writable="true">
       <file filename="example_data" offset="none"/>
       <hash value="16#3466ddf188d8d88cef240e0f02dedb3c09d5a21d6c27b3f3299b74dcd3e30393#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#3000#" type="subject_binary" virtualAddress="
     16#0020_6000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="idle" profile="native">
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
       <file filename="idle_text" offset="none"/>
       <hash value="16#3f85e2e49adb66104e3292d306517c5581cd7c226ba60e66a4414269083f8e8d#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
     virtualAddress="16#0020_1000#" writable="false">
       <file filename="idle_rodata" offset="none"/>
       <hash value="16#db97c62b590d580647fe04fcc6c8a962697fa51f0a7ab475a16967e29cbb4cb9#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="isolation_tests_monitor" profile="native">
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <memory>
       <memory executable="false" logical="result_state" size="16#1000#" virtualAddress="16#0100_0000
     #" writable="true"/>
      </memory>
      <events>
       <source>
        <event id="1" logical="resume_tests"/>
       </source>
       <target>
        <event logical="trap_to_monitor"/>
       </target>
      </events>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
       <file filename="isolation_tests_monitor_text" offset="none"/>
       <hash value="16#7feb0756df521a4a652c42169ef4d4f0c665aef9bb29c09f5f55969baf718fa7#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
     virtualAddress="16#0020_1000#" writable="false">
       <file filename="isolation_tests_monitor_rodata" offset="none"/>
       <hash value="16#dd605fb441991ff03476a906d81d12420eeeb511a501874bdcda0a761a311657#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
     "16#0020_2000#" writable="true">
       <file filename="isolation_tests_monitor_data" offset="none"/>
       <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
      </memory>
```

```
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress="16#1000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
    </provides>
  </component>
  <component name="isolation_tests" profile="native">
    <config>
      <integer name="log_entry_max" value="128"/>
      <integer name="log_buffer_size" value="65535"/>
    </config>
    <depends>
      <library ref="libmudebuglog"/>
    </depends>
    <requires>
      <vcpu>
        <msrs>
          <msr end="16#0174#" mode="r" start="16#0174#"/>
        </msrs>
        <registers>
          <gpr>
            <rip>16#0020_0000#</rip>
          </gpr>
        </registers>
      </vcpu>
      <memory>
        <memory executable="false" logical="read_only" size="16#1000#" virtualAddress="16#1000_0000#" writable="false"/>
        <memory executable="false" logical="result_state" size="16#1000#" virtualAddress="16#0100_0000#" writable="true"/>
      </memory>
      <events>
        <target>
          <event logical="resume_tests"/>
        </target>
      </events>
    </requires>
    <provides>
      <memory executable="true" logical="text" size="16#7000#" type="subject_binary" virtualAddress="16#0020_0000#" writable="false">
        <file filename="isolation_tests_text" offset="none"/>
        <hash value="16#1de77853ba33575f9e137963ff8e14b6d6ae62aac3d5026ae3e5aa016cecf5f9#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#3000#" type="subject_binary" virtualAddress="16#0020_7000#" writable="false">
        <file filename="isolation_tests_rodata" offset="none"/>
        <hash value="16#2c6e5333ad82bfdf14837182f790e8994c8cd3327a12944a3f885c0d0e558fa3#"/>
      </memory>
      <memory executable="false" logical="data" size="16#b000#" type="subject_binary" virtualAddress="16#0020_a000#" writable="true">
        <file filename="isolation_tests_data" offset="none"/>
        <hash value="16#731b8abc3033235953e353ac5e0d68f1611e9739edb0e1ef274a0403ce883bef#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#0001_1000#" type="subject_binary" virtualAddress="16#0021_5000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress="16#1000#" writable="true">
        <fill pattern="16#00#"/>
      </memory>
    </provides>
  </component>
  <component name="linux" profile="linux">
    <requires>
      <memory>
        <memory executable="true" logical="lowmem" size="16#0008_0000#" virtualAddress="16#0002_0000#" writable="true"/>
        <memory executable="true" logical="ram" size="16#1000_0000#" virtualAddress="16#0100_0000#" writable="true"/>
      </memory>
    </requires>
    <provides>
      <memory executable="true" logical="binary" size="16#0078_5000#" type="subject_binary" virtualAddress="16#0040_0000#" writable="true">
        <file filename="bzImage" offset="none"/>
      </memory>
      <memory executable="false" logical="modules_initramfs" size="16#0002_3000#" type="subject_initrd" virtualAddress="16#7113_0000#" writable="false">
        <file filename="modules_initramfs.cpio.gz" offset="none"/>
      </memory>
    </provides>
  </component>
  <component name="ps2_drv" profile="native">
    <depends>
      <library ref="libmudebuglog"/>
    </depends>
```

```
      <requires>
1589   <vcpu>
        <registers>
1591     <gpr>
          <rip>16#0020_0000#</rip>
1593     </gpr>
        </registers>
1595   </vcpu>
       <channels>
1597    <writer event="16" logical="input_events" size="16#1000#" virtualAddress="16#0001_0000_0000#"/
      >
       </channels>
1599   <devices>
        <device logical="ps2">
1601     <irq logical="kbd_irq" vector="49"/>
         <irq logical="mouse_irq" vector="60"/>
1603     <ioPort end="16#0060#" logical="port_60" start="16#0060#"/>
         <ioPort end="16#0064#" logical="port_64" start="16#0064#"/>
1605    </device>
       </devices>
1607  </requires>
      <provides>
1609   <memory executable="false" logical="interrupt_stack" size="16#2000#" type="subject_binary"
      virtualAddress="16#0001_0000#" writable="true">
         <fill pattern="16#00#"/>
1611   </memory>
       <memory executable="true" logical="text" size="16#4000#" type="subject_binary" virtualAddress="
      16#0020_0000#" writable="false">
1613     <file filename="ps2_drv_text" offset="none"/>
         <hash value="16#f7a71b8b5711d460baf3a73a7e41c66759627598737f2d9e2ffb740187a1f40b#"/>
1615   </memory>
       <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
      virtualAddress="16#0020_4000#" writable="false">
1617     <file filename="ps2_drv_rodata" offset="none"/>
         <hash value="16#690afe132af76f72b625a0399814f37c750ecd7c333d2a05f36a48535d23c94c#"/>
1619   </memory>
       <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
      "16#0020_5000#" writable="true">
1621     <file filename="ps2_drv_data" offset="none"/>
         <hash value="16#3588778c9b095a6ec7a846d4bf2a083ed54e0c7076ded731568c6a8cbf751f4b#"/>
1623   </memory>
       <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress="
      16#0020_6000#" writable="true">
1625     <fill pattern="16#00#"/>
       </memory>
1627   <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
      ="16#1000#" writable="true">
         <fill pattern="16#00#"/>
1629   </memory>
      </provides>
1631 </component>
     <component name="sl" profile="native">
1633  <depends>
       <library ref="libmuinit"/>
1635  </depends>
      <requires>
1637   <vcpu>
        <registers>
1639     <gpr>
          <rip>16#0020_0000#</rip>
1641     </gpr>
        </registers>
1643   </vcpu>
       <events>
1645    <source>
          <event id="0" logical="start"/>
1647    </source>
       <target>
1649     <event logical="handle_reset"/>
       </target>
1651   </events>
      </requires>
1653  <provides>
       <memory executable="true" logical="text" size="16#3000#" type="subject_binary" virtualAddress="
      16#0020_0000#" writable="false">
1655     <file filename="sl_text" offset="none"/>
         <hash value="16#e58bf50d4bf40c93e3e55fcbe1a463faf0d7e878a45ca46e7dc9c232e7bda20d#"/>
1657   </memory>
       <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
      virtualAddress="16#0020_3000#" writable="false">
1659     <file filename="sl_rodata" offset="none"/>
         <hash value="16#680bc1a5bea402fb9232ce73620ece595c32696b86c6b9c4f3eb159cbd270e31#"/>
1661   </memory>
       <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
      "16#0020_4000#" writable="true">
1663     <file filename="sl_data" offset="none"/>
         <hash value="none"/>
```

```
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="sm" profile="native">
     <config>
      <boolean name="debug_wrmsr" value="false"/>
      <boolean name="pciconf_emulation_enabled" value="true"/>
      <boolean name="debug_rdtsc" value="false"/>
      <boolean name="debug_rdmsr" value="false"/>
      <boolean name="debug_cpuid" value="false"/>
      <boolean name="debug_cr" value="false"/>
      <boolean name="debug_ioport" value="false"/>
      <boolean name="debug_ept" value="false"/>
     </config>
     <depends>
      <library ref="libmutime"/>
      <library ref="libmudebuglog"/>
      <library ref="libmudm"/>
      <library ref="muinit"/>
     </depends>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <events>
       <source>
        <event id="4" logical="resume_subject"/>
       </source>
       <target>
        <event logical="handle_subject_trap"/>
       </target>
      </events>
     </requires>
     <provides>
      <memory executable="false" logical="interrupt_stack" size="16#2000#" type="subject_binary"
     virtualAddress="16#0001_0000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="true" logical="text" size="16#6000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
       <file filename="sm_text" offset="none"/>
       <hash value="16#ffb98715e613cb0e9bd2363225f6446c5c99cd1d16e4d4baa80ff06a66bcffd4#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
     virtualAddress="16#0020_6000#" writable="false">
       <file filename="sm_rodata" offset="none"/>
       <hash value="16#270ed58f917c50ceadafa64243d327fa49e0ca5c369465c171738dec6ccb3d67#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
     "16#0020_7000#" writable="true">
       <file filename="sm_data" offset="none"/>
       <hash value="16#1645d0c24ddf2415497e7fbfea90b2c0eed2d82fa7d3f305d2112d88f5d38cc3#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress="
     16#0020_8000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="time" profile="native">
     <depends>
      <library ref="libmudebuglog"/>
      <library ref="libmucontrol"/>
     </depends>
     <requires>
      <vcpu>
       <vmx>
        <controls>
         <proc>
          <RDTSCExiting>0<!--
           This is an example of a component that customizes the vCPU
           settings. In this case, direct access to the Time-Stamp Counter
           (TSC) is enabled. The settings made here are merged with the
           (default) values defined by the component profile during policy
           expansion by the Mucfgexpand tool.
```

```
1745            --></RDTSCExiting>
          </proc>
1747       </controls>
         </vmx>
1749       <registers>
          <gpr>
1751        <rip>16#0020_0000#</rip>
          </gpr>
1753       </registers>
        </vcpu>
1755    <channels>
         <array elementSize="16#1000#" logical="export_channels" virtualAddressBase="16#0001_0000_0000#
       ">
1757       <writer logical="time_export1"/>
         </array>
1759    </channels>
        <devices>
1761     <device logical="cmos_rtc">
          <ioPort end="16#0071#" logical="ports" start="16#0070#"/>
1763     </device>
        </devices>
1765   </requires>
       <provides>
1767    <memory executable="true" logical="text" size="16#2000#" type="subject_binary" virtualAddress="
       16#0020_0000#" writable="false">
         <file filename="time_text" offset="none"/>
1769     <hash value="16#d45b62b5f8afa0a838bac83e854996e9a4059f173aac4905265fcfb22fb2f1d0#"/>
        </memory>
1771    <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
       virtualAddress="16#0020_2000#" writable="false">
         <file filename="time_rodata" offset="none"/>
1773     <hash value="16#b0fd849e1cb2a24c297924bebe30b0d11e47258ded43364b726431e322a233b6#"/>
        </memory>
1775    <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
       "16#0020_3000#" writable="true">
         <file filename="time_data" offset="none"/>
1777     <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
        </memory>
1779    <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress="
       16#0020_4000#" writable="true">
         <fill pattern="16#00#"/>
1781    </memory>
        <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
       ="16#1000#" writable="true">
1783     <fill pattern="16#00#"/>
        </memory>
1785   </provides>
      </component>
1787  <component name="vt" profile="native">
       <depends>
1789    <library ref="libmudebuglog"/>
       </depends>
1791   <requires>
        <vcpu>
1793     <registers>
          <gpr>
1795       <rip>16#0020_0000#</rip>
          </gpr>
1797     </registers>
        </vcpu>
1799    <channels>
         <array elementSize="16#0001_0000#" logical="console" vectorBase="64" virtualAddressBase="
       16#0001_0000_0000#">
1801       <reader logical="NIC Linux"/>
           <reader logical="Storage Linux"/>
1803     </array>
         <array elementSize="16#1000#" eventBase="16" logical="input_devices" virtualAddressBase="
       16#0001_0002_0000#">
1805       <writer logical="input_device_1"/>
           <writer logical="input_device_2"/>
1807     </array>
         <reader logical="input_events" size="16#1000#" vector="66" virtualAddress="16#0001_0002_2000#"
       />
1809    </channels>
        <devices>
1811     <device logical="vga">
          <memory executable="false" logical="buffer" size="16#0002_0000#" virtualAddress="16#000a_0000
       #" writable="true"/>
1813       <ioPort end="16#03df#" logical="ports" start="16#03c0#"/>
         </device>
1815    </devices>
        <events>
1817     <source>
          <event id="1" logical="request_reset_slot_1"/>
1819      <event id="30" logical="shutdown">
           <system_poweroff/>
1821      </event>
```

```
        <event id="31" logical="reboot">
         <system_reboot/>
        </event>
       </source>
      </events>
     </requires>
     <provides>
      <memory executable="false" logical="interrupt_stack" size="16#2000#" type="subject_binary"
     virtualAddress="16#0001_0000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="true" logical="text" size="16#b000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
       <file filename="vt_text" offset="none"/>
       <hash value="16#15a0c09815f5f8627aae85f9347b7ef63adf85586d651b427326bfb46016d776#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#3000#" type="subject_binary"
     virtualAddress="16#0020_b000#" writable="false">
       <file filename="vt_rodata" offset="none"/>
       <hash value="16#4d2bd9113c31f1d14351922d3e239ca6453bbcc495a79877f393fc336b6577e8#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress=
     "16#0020_e000#" writable="true">
       <file filename="vt_data" offset="none"/>
       <hash value="16#03ba720e717d15d7b0fe0e0267036398e09ddbfa3d66a38d1455cad48869e39c#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress="
     16#0020_f000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#2000#" type="subject_binary" virtualAddress
     ="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
   </components>
   <subjects>
    <!--
     The `subjects` element holds a list of subjects.
    -->
    <subject name="vt">
     <!--
      A subject is an instance of a component, i.e. an active component in the
      system policy that may be scheduled. Its specification references a
      component and maps all requested logical resources to physical resources
      provided by the system. Additional resources to the ones requested by
      the component can be specified here. This enables specialization of the
      base component specification.
     -->
     <vcpu>
      <vmx>
       <controls>
        <proc>
         <!-- VM-Exit on HLT instruction -->
         <HLTExiting>1</HLTExiting>
        </proc>
       </controls>
      </vmx>
     </vcpu>
     <events>
      <!--
       The subject `events` element specifies all events originating from or
       directed at this subject. The physical attribute is the name of a event
       defined in the global events section.
      -->
      <source>
       <!--
        The event `source` element specifies events that are allowed to be
        triggered by the associated subject.

        The `vmx_exit` group is translated to a lookup table for handling VMX
        exit traps as defined by Intel SDM Vol. 3D, "Appendix C VMX Basic Exit
        Reasons". The `vmcall` group on the other hand is translated into a
        lookup table to handle hypercalls.
       -->
       <group name="vmx_exit">
        <default physical="system_panic">
         <!--
          The `default` element entry can be used to specify an event which should
          be added for all event ids that have not been explicitly specified.
         -->
         <system_panic/>
        </default>
        <!-- Exit Reason 12: HLT -->
        <event id="12" logical="sleep" physical="subject_sleep">
         <subject_sleep/>
```

```
        </event>
       </group>
      </source>
     </events>
     <component ref="vt">
      <!--
       The `component` reference element specifies which component this subject
       instantiates. All logical resources required by the component must be
       mapped to physical resources of the appropriate type. Validators make
       sure that all requirements are satisfied and that no mapping has been
       omitted.
      -->
      <map logical="NIC Linux" physical="virtual_console_1"/>
      <map logical="Storage Linux" physical="virtual_console_2"/>
      <map logical="input_events" physical="input_events"/>
      <map logical="input_device_1" physical="virtual_input_1"/>
      <map logical="input_device_2" physical="virtual_input_2"/>
      <map logical="debuglog" physical="debuglog_subject1"/>
      <map logical="vga" physical="vga">
       <map logical="buffer" physical="buffer"/>
       <map logical="ports" physical="ports">
        <!--
         The `map` element maps a physical resource provided by the system with a
         resource requested by the referenced component.

         This element allows recursion to map child resources as well (e.g.
         device memory, I/O ports etc).
        -->
       </map>
      </map>
      <map logical="request_reset_slot_1" physical="request_reset_slot_1"/>
      <map logical="shutdown" physical="system_poweroff"/>
      <map logical="reboot" physical="system_reboot"/>
     </component>
    </subject>
    <subject name="nic_sm">
     <memory>
      <memory executable="false" logical="status_linux" physical="status_linux_1" virtualAddress="
     16#0200_0000#" writable="false"/>
     </memory>
     <events>
      <source>
       <group name="vmx_exit">
        <default physical="system_panic">
         <system_panic/>
        </default>
       </group>
       <group name="vmcall">
        <event id="0" logical="serial_irq4" physical="serial_irq4_linux_1">
         <!--
          A source `event` entry specifies a source event node, i.e. it registers
          a handler for the given event `id`. These IDs, depending on the event
          group, are either hypercall numbers or VMX basic exit reasons. The valid
          ID ranges of the respective groups are:

          vmx_exit
          0 .. 59

          vmcall
          0 .. 63

          Additionally, the following IDs in `vmx_exit` group are reserved and may
          not be used:

          -   Used by kernel: 1, 7, 41, 52, 55

          -   Reserved by Intel: 35, 38, 42

          It is possible to assign event actions to event source entries.
          Currently supported source event actions are `subject_sleep`,
          `subject_yield`, `unmask_irq`, `system_reboot`, `system_poweroff` and
          `system_panic`, which all have the kernel itself as endpoint.
         -->
        </event>
        <event id="1" logical="reset_linux" physical="reset_linux_1"/>
        <event id="2" logical="load_linux" physical="load_linux_1"/>
       </group>
      </source>
      <target>
       <!--
        The event `target` element specifies events that the subject is an
        *endpoint* of.
       -->
       <event logical="resume_after_load" physical="start_linux_1">
        <!--
         The `event` element in the target section specifies one event endpoint
         by referencing a physical event and assigning a logical name to it.
```

```
        -->
       </event>
       <event id="63" logical="reset" physical="reset_sm_1">
        <reset/>
       </event>
      </target>
     </events>
     <monitor>
      <!--
       The monitor abstraction enables subjects to request access to certain
       data of another subject specified by name. Possible child elements are:

       -   State

       -   Timed_Events

       -   Interrupts

       -   Loader

       See the Muen Component Specification document for details about these
       subject monitor interfaces.
      -->
      <state logical="monitor_state" subject="nic_linux" virtualAddress="16#001e_0000#" writable="
     true"/>
      <loader logical="reload" subject="nic_sm" virtualAddress="16#0000#">
       <!--
        The `loader` mechanism effectively puts the loaded subject denoted by
        the `subject` attribute under loader control, as it is not able to start
        without the help of the loader.

        In more detail, the `loader` monitor element instructs the expander tool
        to map all memory regions of the referenced subject into the address
        space of the monitor subject, using the specified `virtualAddress` as
        offset in the address space of the loader.

        If a memory region of the loaded subject is writable and file-backed,
        the region is replaced with an empty region and linked via the `hashRef`
        mechanism to the original region which is mapped into the loader.

        The state of the loaded subject is then invalidated by clearing the
        `CR4.VMXE` bit in the initial subject CR4 register value. If such a
        subject is scheduled by the kernel, a VMX exit *VM-entry failure due to
        invalid guest state* (33) occurs. See Intel SDM Vol. 3C, "23.7 Enabling
        and Entering VMX Operation" and Intel SDM Vol. 3C, "23.8 Restrictions on
        VMX Operation" for more details. This trap is linked to the loader via
        normal VMX event handling. After handover, the loader initializes the
        memory regions replaced by the expander with the designated content.

        All information required to *load* the loaded subject is provided to the
        loader subject via its own sinfo API. Memory regions prefixed with
        `monitor_sinfo_` provide access to the sinfo regions of the loaded
        subjects. Regions prefixed with `monitor_state_` specify memory regions
        containing the subject register state of the loaded subject.

        The difference between the `monitor_sinfo_` memory region address in the
        loader and the address of the `sinfo` memory region in the target sinfo
        information denotes the `virtualAddress` offset attribute of the
        `loader` element in the policy. This information combined is enough to
        fully construct the initial state of the loaded subject, or to reset a
        subject to its initial state on demand.

        The loader may also optionally check the hashes of the restored regions,
        as this information is provided via the sinfo mechanism as well.
       -->
      </loader>
     </monitor>
     <component ref="sm">
      <map logical="time_info" physical="time_info"/>
      <map logical="debuglog" physical="debuglog_subject2"/>
      <map logical="dm_pciconf_req" physical="nic_dm_request"/>
      <map logical="dm_pciconf_res" physical="nic_dm_response"/>
      <map logical="resume_subject" physical="resume_linux_1"/>
      <map logical="handle_subject_trap" physical="trap_to_sm_1"/>
      <map logical="status" physical="status_sm_1"/>
      <map logical="control" physical="control_sm_1"/>
     </component>
    </subject>
    <subject name="storage_sm">
     <events>
      <source>
       <group name="vmcall">
        <event id="0" logical="serial_irq4" physical="serial_irq4_linux_2"/>
        <event id="1" logical="reset_linux" physical="reset_linux_2"/>
       </group>
       <group name="vmx_exit">
        <default physical="system_panic">
```

```
          <system_panic/>
         </default>
        </group>
       </source>
      </events>
      <monitor>
       <state logical="monitor_state" subject="storage_linux" virtualAddress="16#001e_0000#" writable="true"/>
       <loader logical="reload" subject="storage_sm" virtualAddress="16#0000#"/>
      </monitor>
      <component ref="sm">
       <map logical="time_info" physical="time_info"/>
       <map logical="debuglog" physical="debuglog_subject3"/>
       <map logical="dm_pciconf_req" physical="storage_dm_request"/>
       <map logical="dm_pciconf_res" physical="storage_dm_response"/>
       <map logical="resume_subject" physical="resume_linux_2"/>
       <map logical="handle_subject_trap" physical="trap_to_sm_2"/>
       <map logical="status" physical="status_sm_2"/>
       <map logical="control" physical="control_sm_2"/>
      </component>
     </subject>
     <subject name="time">
      <vcpu>
       <vmx>
        <controls>
         <proc>
          <!-- VM-Exit on HLT instruction -->
          <HLTExiting>1</HLTExiting>
         </proc>
        </controls>
       </vmx>
      </vcpu>
      <events>
       <source>
        <group name="vmx_exit">
         <default physical="system_panic">
          <system_panic/>
         </default>
         <!-- Exit Reason 12: HLT -->
         <event id="12" logical="sleep" physical="subject_sleep">
          <subject_sleep/>
         </event>
        </group>
       </source>
      </events>
      <component ref="time">
       <map logical="time_export1" physical="time_info"/>
       <map logical="debuglog" physical="debuglog_subject4"/>
       <map logical="cmos_rtc" physical="cmos_rtc">
        <map logical="ports" physical="ports"/>
       </map>
       <map logical="status" physical="status_time"/>
       <map logical="control" physical="control_time"/>
      </component>
     </subject>
     <subject name="nic_sl">
      <events>
       <source>
        <group name="vmx_exit">
         <default physical="system_panic">
          <system_panic/>
         </default>
        </group>
       </source>
      </events>
      <monitor>
       <loader logical="monitor_loader_nic_linux" subject="nic_linux" virtualAddress="16#0001_0000_0000#"/>
      </monitor>
      <component ref="sl">
       <map logical="start" physical="start_linux_1"/>
       <map logical="handle_reset" physical="load_linux_1"/>
       <map logical="status" physical="status_linux_1"/>
       <map logical="control" physical="control_linux_1"/>
      </component>
     </subject>
     <subject name="ps2">
      <vcpu>
       <vmx>
        <controls>
         <proc>
          <!-- VM-Exit on HLT instruction -->
          <HLTExiting>1</HLTExiting>
         </proc>
        </controls>
       </vmx>
      </vcpu>
```

```
    <events>
2161     <source>
      <group name="vmx_exit">
2163     <default physical="system_panic">
       <system_panic/>
2165     </default>
       <!-- Exit Reason 12: HLT -->
2167     <event id="12" logical="sleep" physical="subject_sleep">
       <subject_sleep/>
2169     </event>
      </group>
2171   </source>
    </events>
2173   <component ref="ps2_drv">
     <map logical="input_events" physical="input_events"/>
2175   <map logical="debuglog" physical="debuglog_subject5"/>
     <map logical="ps2" physical="ps2">
2177   <map logical="kbd_irq" physical="kbd_irq"/>
     <map logical="mouse_irq" physical="mouse_irq"/>
2179   <map logical="port_60" physical="port_60"/>
     <map logical="port_64" physical="port_64"/>
2181   </map>
    </component>
2183  </subject>
   <subject name="example">
2185   <events>
     <source>
2187   <group name="vmx_exit">
       <default physical="system_panic">
2189     <system_panic/>
       </default>
2191   </group>
     </source>
2193   </events>
     <monitor>
2195   <state logical="monitor_state" subject="storage_linux" virtualAddress="16#001e_0000#" writable=
    "false"/>
     <loader logical="reload" subject="example" virtualAddress="16#0000#"/>
2197   </monitor>
     <component ref="example">
2199   <map logical="example_request" physical="example_request"/>
     <map logical="example_response" physical="example_response"/>
2201   <map logical="debuglog" physical="debuglog_example"/>
     <map logical="sleep" physical="subject_sleep"/>
2203   <map logical="yield" physical="subject_yield"/>
     <map logical="timer" physical="example_self"/>
2205   <map logical="inject_timer" physical="example_self"/>
     <map logical="control" physical="control_example"/>
2207   <map logical="status" physical="status_example"/>
     <map logical="filled_region" physical="example_filled_region"/>
2209   </component>
   </subject>
2211  <subject name="controller">
     <vcpu>
2213   <vmx>
       <controls>
2215     <proc>
         <!-- VM-Exit on PAUSE instruction -->
2217     <PAUSEExiting>1</PAUSEExiting>
       </proc>
2219   </controls>
     </vmx>
2221   </vcpu>
     <events>
2223   <source>
       <group name="vmx_exit">
2225     <default physical="system_panic">
         <system_panic/>
2227     </default>
       <!-- Exit Reason 40: PAUSE -->
2229     <event id="40" logical="yield" physical="subject_yield">
         <subject_yield/>
2231     </event>
      </group>
2233   </source>
    </events>
2235   <monitor>
     <interrupts logical="pending_interrupts" subject="controller" virtualAddress="16#0030_0000#"
    writable="true"/>
2237   </monitor>
     <component ref="controller">
2239   <map logical="debuglog" physical="debuglog_controller"/>
     <map logical="control_1" physical="control_time"/>
2241   <map logical="control_2" physical="control_sm_1"/>
     <map logical="control_3" physical="control_sm_2"/>
2243   <map logical="control_4" physical="control_example"/>
     <map logical="control_5" physical="control_linux_1"/>
```

```
      <map logical="status_1" physical="status_time"/>
      <map logical="status_2" physical="status_sm_1"/>
      <map logical="status_3" physical="status_sm_2"/>
      <map logical="status_4" physical="status_example"/>
      <map logical="status_5" physical="status_linux_1"/>
      <map logical="reset_slot_1_sm" physical="reset_sm_1"/>
      <map logical="reset_slot_1_linux" physical="reset_slot_1"/>
      <map logical="request_reset_slot_1" physical="request_reset_slot_1"/>
     </component>
    </subject>
    <subject name="nic_dm">
     <devices>
      <!--
       List of device references. Used to grant a subject access to hardware
       devices and their resources.
      -->
      <device logical="nic" physical="ethernet_controller_1">
       <!--
        The 'device' element allows a subject access to devices referenced via
        the 'physical' attribute.

        For PCI devices only a single virtual bus is provided (bus 0). The 'pci'
        element may be used to place the device at a specific location (BDF). If
        no other logical device resources of the device are specified, then the
        expander tool will map all physical devices resources into the subject.
        When logical device resources are explicitly specified, then only access
        to those are actually granted. The physical attribute must be either a
        reference to an existing physical device, device alias or device class.
        Validators check that this is the case.
       -->
       <pci bus="16#00#" device="16#01#" function="0"/>
       <memory executable="false" logical="mmconf" physical="mmconf" writable="true">
        <!--
         The device 'memory' element maps the device memory region referenced via
         the 'physical' attribute into the subject address space at address
         'virtualAddress'. The 'executable', 'writable' attributes define the
         access permissions for the subject.
        -->
       </memory>
      </device>
     </devices>
     <events>
      <source>
       <group name="vmx_exit">
        <default physical="system_panic">
         <system_panic/>
        </default>
       </group>
      </source>
     </events>
     <component ref="dm">
      <map logical="debuglog" physical="debuglog_subject6"/>
      <map logical="request" physical="nic_dm_request"/>
      <map logical="response" physical="nic_dm_response"/>
     </component>
    </subject>
    <subject name="storage_dm">
     <events>
      <source>
       <group name="vmx_exit">
        <default physical="system_panic">
         <system_panic/>
        </default>
       </group>
      </source>
     </events>
     <component ref="dm">
      <map logical="debuglog" physical="debuglog_subject7"/>
      <map logical="request" physical="storage_dm_request"/>
      <map logical="response" physical="storage_dm_response"/>
     </component>
    </subject>
    <subject name="dbgserver">
     <events>
      <source>
       <group name="vmx_exit">
        <default physical="system_panic">
         <system_panic/>
        </default>
       </group>
      </source>
     </events>
     <component ref="dbgserver">
      <map logical="log_channel1" physical="debuglog_subject1"/>
      <map logical="log_channel2" physical="debuglog_subject2"/>
      <map logical="log_channel3" physical="debuglog_subject3"/>
      <map logical="log_channel4" physical="debuglog_subject4"/>
```

```
      <map logical="log_channel5" physical="debuglog_subject5"/>
      <map logical="log_channel_example" physical="debuglog_example">
      </map>
      <map logical="log_channel_6" physical="debuglog_controller"/>
      <map logical="log_channel7" physical="debuglog_subject6"/>
      <map logical="log_channel8" physical="debuglog_subject7"/>
      <map logical="crash_audit" physical="crash_audit"/>
      <map logical="debugconsole" physical="serial_device_1">
       <map logical="port" physical="ioport1"/>
      </map>
      <map logical="reboot" physical="system_reboot"/>
      <map logical="shutdown" physical="system_poweroff"/>
     </component>
    </subject>
    <subject name="nic_linux">
     <bootparams>console=hvc console=ttyS0 hostname=nic_linux</bootparams>
     <memory>
      <memory executable="false" logical="initramfs" physical="initramfs" virtualAddress="16#7000
    _0000#" writable="false"/>
     </memory>
     <devices>
      <device logical="eth0" physical="nic_1">
       <pci bus="16#00#" device="16#01#" function="0"/>
      </device>
      <device logical="aditional_nics" physical="additional_nics"/>
     </devices>
     <events>
      <source>
       <group name="vmx_exit">
        <default physical="trap_to_sm_1"/>
       </group>
       <group name="vmcall">
        <event id="30" logical="reboot" physical="request_reset_slot_1"/>
        <event id="31" logical="timer" physical="timer_linux_1"/>
       </group>
      </source>
      <target>
       <event logical="resume_after_trap" physical="resume_linux_1"/>
       <event id="63" logical="reset" physical="reset_linux_1">
        <reset/>
       </event>
       <event id="62" logical="reset_from_vt" physical="reset_slot_1">
        <reset/>
       </event>
       <event logical="serial_irq4" physical="serial_irq4_linux_1">
        <inject_interrupt vector="52">
         <!--
          Instructs the SK to inject a guest interrupt with given vector on event
          occurrence.
         -->
        </inject_interrupt>
       </event>
       <event logical="timer" physical="timer_linux_1">
        <inject_interrupt vector="236"/>
       </event>
      </target>
     </events>
     <channels>
      <!--
       The 'channel' section of a subject declares references to communication
       channels. The referenced channels become accessible to the requesting
       subject either as reader or writer endpoint.
      -->
      <reader logical="virtual_input" physical="virtual_input_1" vector="64" virtualAddress="16#0001
    _0000_0000#">
       <!--
        A channel 'reader' element references a global communication channel as
        reader endpoint, i.e. the channel is mapped read-only into the subject
        address space.
       -->
      </reader>
      <writer event="16" logical="virtual_console" physical="virtual_console_1" virtualAddress="
    16#0001_0000_1000#">
       <!--
        A channel 'writer' element references a global communication channel as
        writer endpoint, i.e. the channel is mapped with write permissions into
        the subject address space.
       -->
      </writer>
      <reader logical="testchannel_2" physical="testchannel_2" virtualAddress="16#0001_0001_1000#"/>
      <writer logical="testchannel_1" physical="testchannel_1" virtualAddress="16#0001_0001_2000#"/>
      <reader logical="testchannel_4" physical="testchannel_4" virtualAddress="16#0001_0001_3000#"/>
      <writer logical="testchannel_3" physical="testchannel_3" virtualAddress="16#0001_0011_3000#"/>
     </channels>
     <component ref="linux">
      <map logical="lowmem" physical="nic_linux|lowmem"/>
      <map logical="ram" physical="nic_linux|ram"/>
```

```
      </component>
     </subject>
     <subject name="storage_linux">
      <bootparams>console=hvc console=ttyS0 hostname=storage_linux</bootparams>
      <memory>
       <memory executable="false" logical="initramfs" physical="initramfs" virtualAddress="16#7000
      _0000#" writable="false"/>
      </memory>
      <devices>
       <device logical="xhci" physical="usb_controller_1"/>
      </devices>
      <events>
       <source>
        <group name="vmx_exit">
         <default physical="trap_to_sm_2"/>
        </group>
        <group name="vmcall">
         <event id="31" logical="timer" physical="timer_linux_2"/>
        </group>
       </source>
       <target>
        <event logical="resume_after_trap" physical="resume_linux_2"/>
        <event id="63" logical="reset" physical="reset_linux_2">
         <reset/>
        </event>
        <event logical="serial_irq4" physical="serial_irq4_linux_2">
         <inject_interrupt vector="52"/>
        </event>
        <event logical="timer" physical="timer_linux_2">
         <inject_interrupt vector="236"/>
        </event>
       </target>
      </events>
      <channels>
       <reader logical="virtual_input" physical="virtual_input_2" vector="64" virtualAddress="16#0001
      _0000_0000#"/>
       <writer event="16" logical="virtual_console" physical="virtual_console_2" virtualAddress="
      16#0001_0000_1000#"/>
       <reader logical="example_response" physical="example_response" vector="65" virtualAddress="
      16#0001_0001_1000#"/>
       <writer event="17" logical="example_request" physical="example_request" virtualAddress="16#0001
      _0001_2000#"/>
       <reader logical="testchannel_1" physical="testchannel_1" virtualAddress="16#0001_0001_3000#"/>
       <writer logical="testchannel_2" physical="testchannel_2" virtualAddress="16#0001_0001_4000#"/>
       <reader logical="testchannel_3" physical="testchannel_3" virtualAddress="16#0001_0001_5000#"/>
       <writer logical="testchannel_4" physical="testchannel_4" virtualAddress="16#0001_0011_5000#"/>
      </channels>
      <component ref="linux">
       <map logical="lowmem" physical="storage_linux|lowmem"/>
       <map logical="ram" physical="storage_linux|ram"/>
      </component>
     </subject>
    </subjects>
    <scheduling tickRate="100000">
     <!--
      The Muen SK implements a fixed, cyclic scheduler. The `scheduling`
      element is used to specify such a static plan by means of a major frame.
      A major frame consist of an arbitrary number of minor frames. Minor
      frames in turn specify a duration in number of ticks a scheduling
      partition is scheduled.

      Scheduling partitions defined in the `partitions` element consist of one
      or more scheduling groups, which in turn specify one or more subjects to
      be scheduled. *Scheduling groups* are used to define groups of
      cooperating subjects, which are allowed to hand over execution to a
      subject in the same scheduling group. This is done via *handover*
      events. Membership of a scheduling group must be specified explicitly in
      the policy, validators enforce that these settings are correct by
      calculating the chain of handover events.

      While scheduling groups support the efficient cooperation of multiple
      subjects, subjects which need to be spatially but not temporally
      isolated from each other cannot profit from it. To efficiently support
      this use-case, the scheduling partition concept is implemented.

      Within a *scheduling partition*, all scheduling groups are scheduled
      round robin with preemption and the opportunity to yield and/or sleep.
      If a subject in a scheduling group sleeps or yields, the next scheduling
      group in the scheduling partition is scheduled. More precisely: the
      active subject of the next scheduling group is executed by the SK.

      Note that prioritization is not implemented on purpose to avoid any
      starvation issues[^1]. The yield operation maps to the x86_64 `PAUSE`
      instruction, while sleep corresponds to `HLT`. See the *Muen Component
      Specification* document for more information on this topic.

      Minor frames designate the scheduling partition that is to be executed
```

```
    for the given amount of ticks. The scheduling partition attribute `name`
    uniquely identifies a scheduling partition. On first activation, the
    first scheduling group (in XML-order) is scheduled. Within the
    scheduling group, the first subject (again in XML-order) is executed.
    The active subject of a scheduling group may change over time, as the
    cooperating subjects initiate handover events.

    The tickRate attribute of the `scheduling` element has the unit Hertz
    (Hz) and specifies the number of clock ticks per second. The ticks
    attribute of minor frames is expressed in terms of this tick rate. As an
    example: if we want to declare the minor frame duration in terms of
    microseconds (10<sup>6</sup>) then a tick rate of 1000000 must be used.

    The duration of a major frame must be the same on each CPU, meaning the
    sum of all minor frame ticks for any given CPU must be identical.
    However, different major frames can have arbitrary length.

    The Tau0 subject designates to the kernel which major frame is the
    currently active one. At the end of each major frame, the kernel
    determines the active major frame and switches to that scheduling plan
    for the duration of the major frame.

    [^1]: Prioritization with starvation protection cannot be implemented
    with low complexity
   -->
    <partitions>
     <!--
      The `partitions` element is used to specify all scheduling partitions of
      the system.
     -->
     <partition name="nic_linux">
      <!--
       The scheduling `partition` element is used to specify a collection of
       scheduling groups consisting of subjects that require spatial but not
       temporal isolation from each other. Within a scheduling partition, all
       scheduling groups are scheduled round robin with preemption (i.e.
       non-cooperatively) and the opportunity to yield and/or sleep.

       A scheduling partition must contain at least one scheduling group.
      -->
      <group>
       <!--
        The scheduling `group` element is used to specify a collection of
        subjects that may cooperatively schedule each other via handover events.
        Scheduling groups must contain at least one subject. As an example, a
        Linux subject and its associated Subject Monitor (SM), Subject Loader
        (SL) and Device Manager (DM) form a scheduling group.
       -->
       <subject name="nic_linux"/>
       <subject name="nic_sm"/>
       <subject name="nic_sl"/>
       <subject name="nic_dm"/>
      </group>
     </partition>
     <partition name="ps2_driver">
      <group>
       <subject name="ps2"/>
      </group>
     </partition>
     <partition name="controller">
      <group>
       <subject name="controller"/>
      </group>
     </partition>
     <partition name="idle_0">
      <group>
       <subject name="mugenschedcfg_auto_idle_0"/>
      </group>
     </partition>
     <partition name="storage_linux">
      <group>
       <subject name="storage_linux"/>
       <subject name="storage_sm"/>
       <subject name="storage_dm"/>
      </group>
      <group>
       <subject name="example"/>
      </group>
     </partition>
     <partition name="debugserver">
      <group>
       <subject name="time"/>
      </group>
      <group>
       <subject name="dbgserver"/>
      </group>
     </partition>
```

```
     <partition name="vt">
      <group>
       <subject name="vt"/>
      </group>
     </partition>
     <partition name="tau0">
      <group>
       <subject name="tau0"/>
      </group>
     </partition>
     <partition name="idle_1">
      <group>
       <subject name="mugenschedcfg_auto_idle_1"/>
      </group>
     </partition>
    </partitions>
    <majorFrame>
     <!--
      A major frame consists of a sequence of minor frames for a given CPU.
      When the end of a major frame is reached, all CPUs synchronize and the
      scheduler starts over from the beginning using the first minor frame
      again. This means that major frames are repeated in a cyclic fashion
      until a different major frame is designated via the Tau0 interface.
     -->
     <cpu id="0">
      <!--
       The `cpu` element is used to specify major frames for each CPU of the
       system.
      -->
      <minorFrame partition="nic_linux" ticks="4">
       <!--
        A minor frame specifies the number of scheduling ticks a partition is
        allowed to run on the CPU specified by the parent `cpu` element.
       -->
      </minorFrame>
      <minorFrame partition="ps2_driver" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="controller" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
```

```
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
      </cpu>
      <cpu id="1">
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="debugserver" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="15"/>
        <minorFrame partition="debugserver" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="vt" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="vt" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="vt" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="vt" ticks="1"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="tau0" ticks="1"/>
        <minorFrame partition="idle_1" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="idle_1" ticks="1"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="idle_1" ticks="1"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="idle_1" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="idle_1" ticks="1"/>
```

```
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="15"/>
      <minorFrame partition="debugserver" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="vt" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="vt" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="vt" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="vt" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="idle_1" ticks="3"/>
     </cpu>
    </majorFrame>
    <majorFrame>
     <cpu id="0">
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="ps2_driver" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="controller" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
      <minorFrame partition="idle_0" ticks="1"/>
      <minorFrame partition="nic_linux" ticks="4"/>
```

```
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
        <minorFrame partition="nic_linux" ticks="4"/>
        <minorFrame partition="idle_0" ticks="1"/>
      </cpu>
      <cpu id="1">
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="debugserver" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="vt" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="storage_linux" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="idle_1" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="idle_1" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="idle_1" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="idle_1" ticks="2"/>
        <minorFrame partition="storage_linux" ticks="3"/>
        <minorFrame partition="debugserver" ticks="2"/>
```

```
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="1"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="vt" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="1"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="vt" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="1"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="vt" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="1"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="vt" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="debugserver" ticks="1"/>
      <minorFrame partition="idle_1" ticks="1"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="vt" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="storage_linux" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
      <minorFrame partition="storage_linux" ticks="3"/>
      <minorFrame partition="idle_1" ticks="2"/>
    </cpu>
  </majorFrame>
 </scheduling>
</system>
```

Listing 8.1: Demo System (VT-d)

Chapter 9

Bibliography

[1] Adrian-Ken Rueegsegger and Reto Buerki. *Muen Component Specification*.

[2] Adrian-Ken Rueegsegger and Reto Buerki. *Muen Separation Kernel*.