
# Core Information Model (CoreModel)

## TR-512.12

## Software

Version 1.5
September 2021

ONF Document Type: Technical Recommendation
ONF Document Name: Core Information Model version 1.5

## Disclaimer



Open Networking Foundation
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
www.opennetworking.org



## Important note

This Technical Recommendations has been approved by the Project TST, but has not been approved by the ONF board. This Technical Recommendation is an update to a previously released TR specification, but it has been approved under the ONF publishing guidelines for 'Informational' publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents. The designation of '-info' at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

# Table of Contents



# List of Figures



# Document History

| Version | Date | Description of Change |
|---|---|---|
| 1.4 | November 2018 | Version 1.4 (Initial Version) |
| 1.5 | September 2021 | Enhancements to model structure. |

# 1 Introduction to the document suite

This document is an addendum to the TR-512 ONF Core Information Model and forms part of the description of the ONF-CIM. For general overview material and references to the other parts refer to TR-512.1.

## 1.1 References

For a full list of references see TR-512.1.

## 1.2 Definitions

For a full list of definition see TR-512.1.

## 1.3 Conventions

See TR-512.1 for an explanation of:

- UML conventions
- Lifecycle Stereotypes
- Diagram symbol set

## 1.4 Viewing UML diagrams

Some of the UML diagrams are very dense. To view them either zoom (sometimes to 400%), open the associated image file (and zoom appropriately) or open the corresponding UML diagram via Papyrus (for each figure with a UML diagram the UML model diagram name is provided under the figure or within the figure).

## 1.5 Understanding the figures

Figures showing fragments of the model using standard UML symbols as well as figures illustrating application of the model are provided throughout this document. Many of the application-oriented figures also provide UML class diagrams for the corresponding model fragments (see TR-512.1 for diagram symbol sets). All UML diagrams depict a subset of the relationships between the classes, such as inheritance (i.e. specialization), association relationships (such as aggregation and composition), and conditional features or capabilities. Some UML diagrams also show further details of the individual classes, such as their attributes and the data types used by the attributes.

# 2 Introduction to Software

The focus of this document is on the software aspects of network devices and compute hosts, and can be split into two broad areas:

1. Software inventory (similar to hardware inventory)
2. Software functionality (equivalent to 'running hardware', which isn't explicitly modelled in the ONF CIM)

Note that we want to reuse the existing functionality classes, ProcessingConstruct and ConstraintDomain because we want to decouple the provided functionality from how it is implemented (in hardware only, in software only or a combination of both hardware and software).

Another challenge is to link everything together, so that the model can:

- Show how running software provides functionality (similar to running hardware)
- Support management of memory, CPU and storage capacity (related back to its usage by running software)
- Show how the combination of hardware and software together produces functionality
- Consistently represent software running directly on hardware CPU and memory as well as the VMM/VM and container cases

Note that there are a number of scenarios that the software model should cover, including:

- Hypervisor/VMM as running software hosts VMs
- VM as running software has guest operating system
- operating system as running software enables running applications
- VM Image is installed element
- (Linux) container as running software enables running applications
- Software agent as running software
- Container engine as running software hosts containers
- Container as running software enables running applications
- Container image is installed element

A lot of research found only one paper that has a model useful for this subject [Fernandez], which was used as the starting point for the VM part of the model.

Note that the TR-512 model:

- Also support a bare metal hypervisor (in effect by adding an association from Hardware to VMM in the diagram below).
- Doesn't include VM Images (VMI) or a repository for the images.

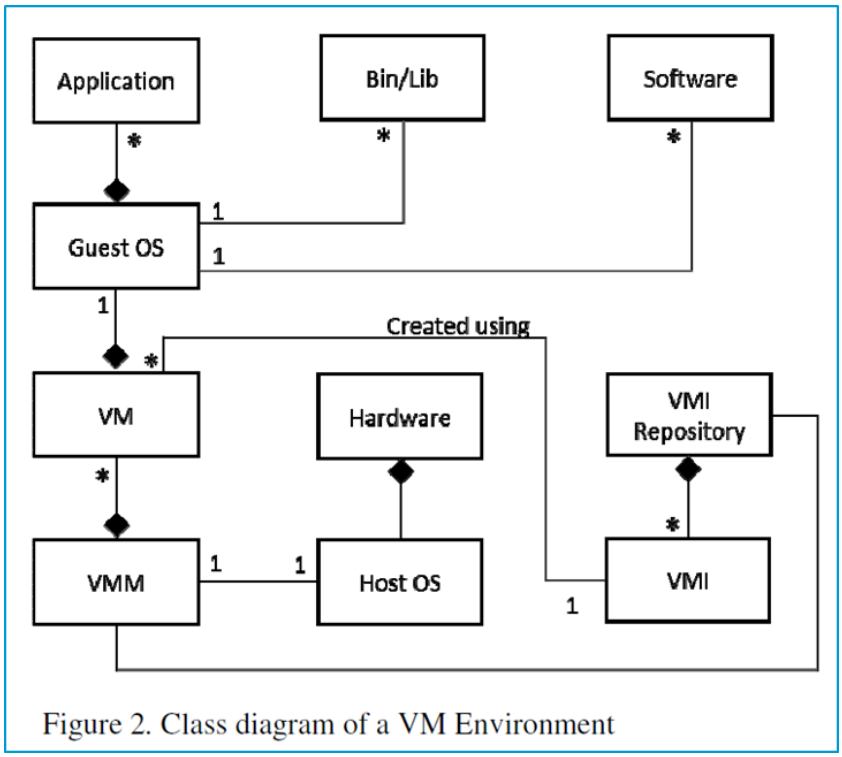



**Figure 2-1 – Hosted VM Model from [Fernandez]**

A data dictionary that sets out the details of all classes, data types and attributes is also provided (TR-512.DD).

# 3 Purpose and essentials of the Software Model

## 3.1 Background

The ONF CIM currently represents all types of functionality, but only relates hardware provided functionality back to its providing implementation. The aim of this module is to provide a similar representation for functionality that is provided by software. This allows the representation of not only the functionality, but also how it is implemented. For example, a BGP ProcessingConstruct instance may require a separate BGP software process, and that running software process may consume a certain amount of CPU and memory. The software model will tie this all together, to enable management of CPU and memory capacity and also how functionality is provided.

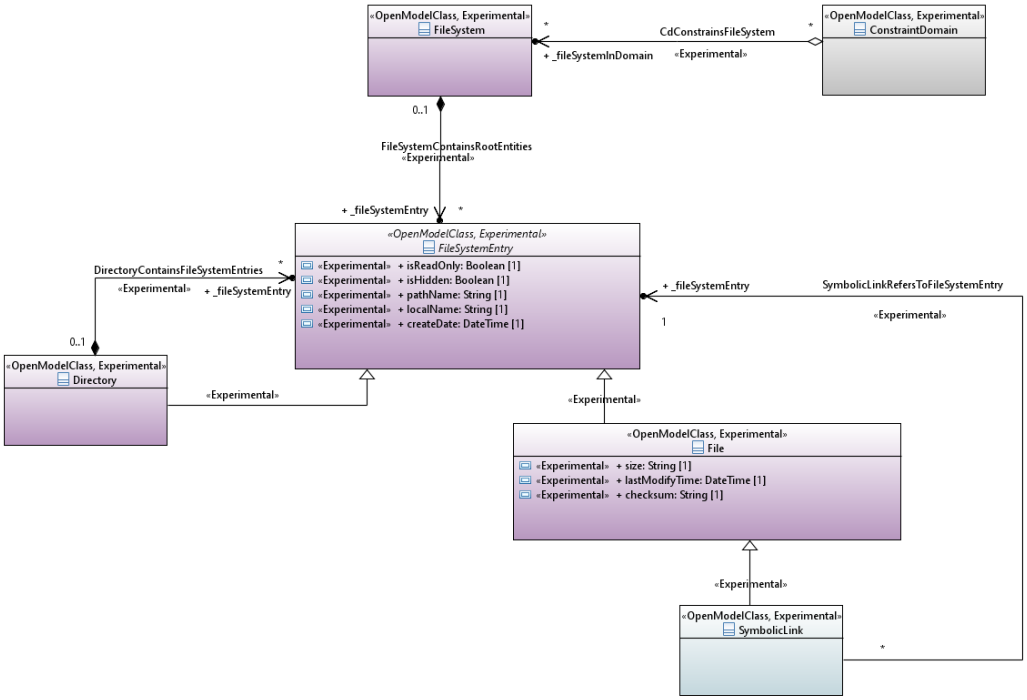
## 3.2 Model

Representing the software inventory requires us to show where the software is stored. To do this we will also define a file system model.

### 3.2.1 File System Model

A FileSystem contains FileSystemEntries which can either be Files or Directories. Directories can contain Files and Directories. This model contains the key attributes only and can be extended in the future if required.

Note that this document does not define a storage model.




CoreModel diagram: Software-FileSystem

**Figure 3-1 Skeleton Class Diagram of key object classes**

#### 3.2.1.1 Directory

Qualified Name: CoreModel::CoreSoftwareModel::FileSystem::Directory

A Directory is a collection of Files and other Directories. Because a Directory can contain other Directories, this allows a hierarchical file system to be represented.

Inherits properties from:

- FileSystemEntry

This class is Experimental.

Table 1: Attributes for Directory

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _fileSystemEntry | Experimental | Directory entry, which can be a File or another Directory. |

#### 3.2.1.2 File

Qualified Name: CoreModel::CoreSoftwareModel::FileSystem::File

A File is a data structure used to store information (user data, commands, software etc.) in a computer system. Note that in this CIM class, we are only storing the metadata associated with the File, not the actual contents of the File.

Inherits properties from:

- FileSystemEntry

This class is Experimental.

Table 2: Attributes for File

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| size | Experimental | The size of the File, in bytes. |
| lastModifyTime | Experimental | The date time that the File was last modified. |
| checksum | Experimental | A checksum that can be used to validate the contents of the File (in case of corruption or malicious changes) using a hashing algorithm like MD5 or SHA1. |

#### 3.2.1.3 FileSystem

Qualified Name: CoreModel::CoreSoftwareModel::FileSystem::FileSystem

A FileSystem organizes the data on a storage system so that it can be easily created, updated and accessed. It manages the Directories and also the metadata for the Files.
This class is Experimental.

Table 3: Attributes for FileSystem

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _fileSystemEntry | Experimental | The root FileSystem entries, which can be Files or Directories. |

#### 3.2.1.4 FileSystemEntry

Qualified Name: CoreModel::CoreSoftwareModel::FileSystem::FileSystemEntry

FileSystemEntry is an abstraction of File and Directory, useful when there is a need to reference both classes. It also allows for an easy representation of a hierarchical filesystem.

This class is abstract.

This class is Experimental.

Table 4: Attributes for FileSystemEntry

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| isReadOnly | Experimental | If the File contents can be modified or not. |
| isHidden | Experimental | If the File is hidden by the FileSystem. |
| pathName | Experimental | The full pathname of the entry, back to the root Directory. |
| localName | Experimental | The name of the entry in its Directory. |
| createDate | Experimental | The date that the entry was created. |

#### 3.2.1.5 SymbolicLink

Qualified Name: CoreModel::CoreSoftwareModel::FileSystem::SymbolicLink

A SymbolicLink is a File that contains a path reference to a File or a Directory.

Inherits properties from:

- File

This class is Experimental.

Table 5: Attributes for SymbolicLink

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _fileSystemEntry | Experimental | The FileSystemEntry that this SymbolicLink refers to. |

### 3.2.2 Core Software Model

InstalledSoftwareComponent records the software artifacts available to be run and their location in the FileSystem.

RunningSoftwareProcess is an abstract class that represents all currently running software processes.

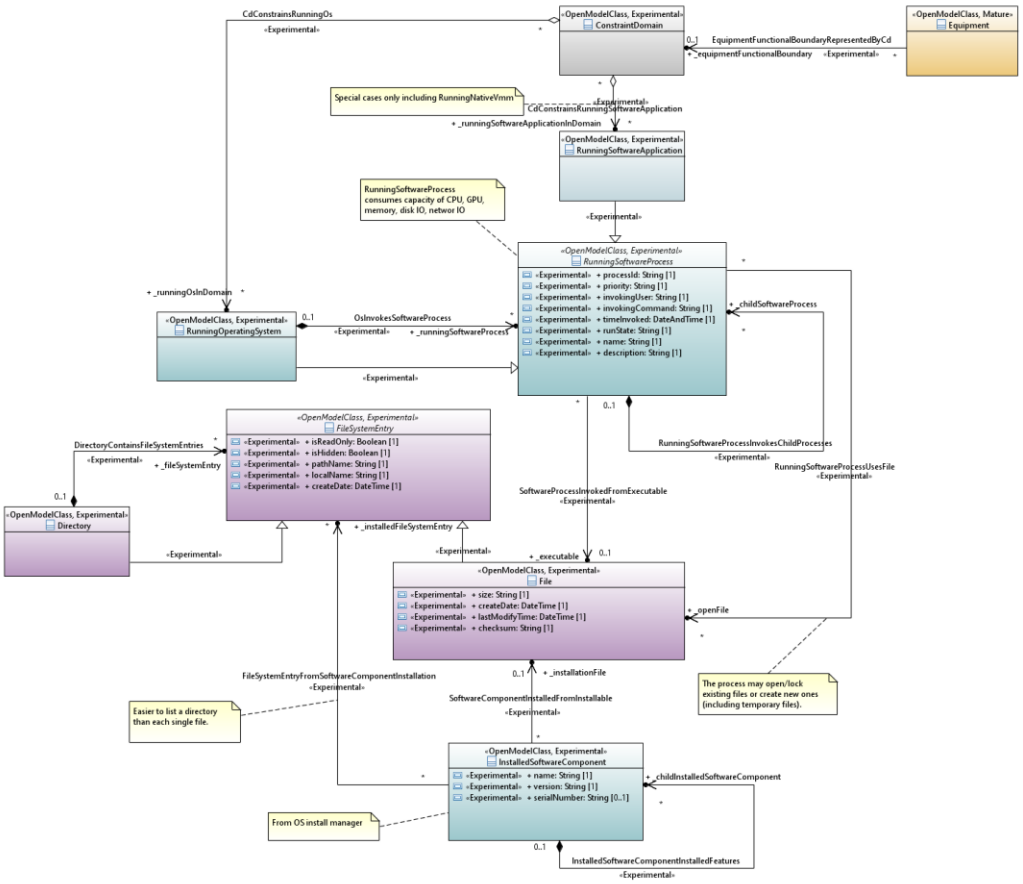
RunningSoftwareApplication is the general case to be used except for the special OS, VMM/VM and container cases which have their own concrete subclasses.

RunningOperatingSystem is recorded as a special case of RunningSoftwareProcess (that other software processes can run in), so there are separate associations for these two cases.

Usually only the RunningOperatingSystem would be attached to a ConstraintDomain, but there may be other cases where a RunningSoftwareApplication to ConstraintDomain association instance may be required (such as distributed software processes).

The FileSystem model relates the RunningSoftwareProcesses to the InstalledSoftwareComponents. If this information is not available (and hence the FileSystem part of the model is not populated) then the model can still be used, but the linkage between installed and running software won't be available.




CoreModel diagram: Software-RunningSoftware

**Figure 3-2 Skeleton Class Diagram of key object classes**

#### 3.2.2.1 InstalledSoftwareComponent

Qualified Name:
CoreModel::CoreSoftwareModel::RunningSoftware::InstalledSoftwareComponent

InstalledSoftwareComponent is part of the software inventory. It represents an application or an application suite or an application (feature) option. It is the unit of installation. Note that this is operating system dependent. For example, Microsoft DOS 3.3 didn't have an installation manager.

This class is Experimental.

Table 6: Attributes for InstalledSoftwareComponent

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| name | Experimental | The name of the installed component as returned by the operating system. |
| version | Experimental | The version of the installed component as returned by the operating system. |
| serialNumber | Experimental | As part of software licensing, a serial number may be available for the installation. |
| _installationFile | Experimental | If available, the File that the installation was from. |
| _childInstalledSoftwareComponent | Experimental | References to any child installations. |
| _installedFileSystemEntry | Experimental | References to any Directories or Files created or used by the installation. Note that installations may share files. So installation 1 may create key.dll in a common area, and installation 2 would create this if it wasn't present. Because it is already present, installation 2 just references the file. Now if installation 1 is uninstalled, key.dll is not removed as there is still a reference to it. |

#### 3.2.2.2 RunningOperatingSystem

Qualified Name: CoreModel::CoreSoftwareModel::RunningSoftware::RunningOperatingSystem

An operating system is a running software process that enables software applications to utilize the computer's hardware.

Inherits properties from:

- RunningSoftwareProcess

This class is Experimental.

Table 7: Attributes for RunningOperatingSystem

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _runningSoftwareProcess | Experimental | The software processes running under this operating system. |
| _runningContainerEngine | Experimental | The running container engines supported by the running operating system. |
| _runningHostOsVmm | Experimental | The host OS VMMs running under this operating system. |

#### 3.2.2.3 RunningSoftwareApplication

Qualified Name:
CoreModel::CoreSoftwareModel::RunningSoftware::RunningSoftwareApplication

This covers generic software processes that don't have a special subclass (operating system, virtual machine and container special cases have their own specific subclasses).

Inherits properties from:

- RunningSoftwareProcess

This class is Experimental.

#### 3.2.2.4 RunningSoftwareProcess

Qualified Name: CoreModel::CoreSoftwareModel::RunningSoftware::RunningSoftwareProcess

This is the unit of software execution. It could be a running application or a thread (the smallest level of software execution).
This class is abstract.

This class is Experimental.

Table 8: Attributes for RunningSoftwareProcess

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| processId | Experimental | The identifier provided by the operating system. |
| priority | Experimental | The process priority which is used by a multi-tasking operating system to assign resource allocations for the different running software processes. |
| invokingUser | Experimental | The username of the account that invoked the process. |
| invokingCommand | Experimental | The command string that invoked the process. |
| timeInvoked | Experimental | The date time when the process was invoked. |
| runState | Experimental | The current run state. A software process may be actively running or suspended (or some other state supported by the operating system). |
| name | Experimental | The name of the process as assigned by the operating system. |
| description | Experimental | The description of the process as assigned by the operating system. |
| _executable | | A reference to the executable file (invoked via the invokingCommand). Note that the invoking command may not list the full file path. |
| _childSoftwareProcess | Experimental | References to a software process's subprocesses and threads. |
| _openFile | Experimental | References to any files opened and/or locked by the running process. |

### 3.2.3 Virtual Machine Model

A VirtualMachineMonitor (VMM) or Hypervisor, is a type of software process that can run VirtualMachines (VM).

There are two types of VMMs :

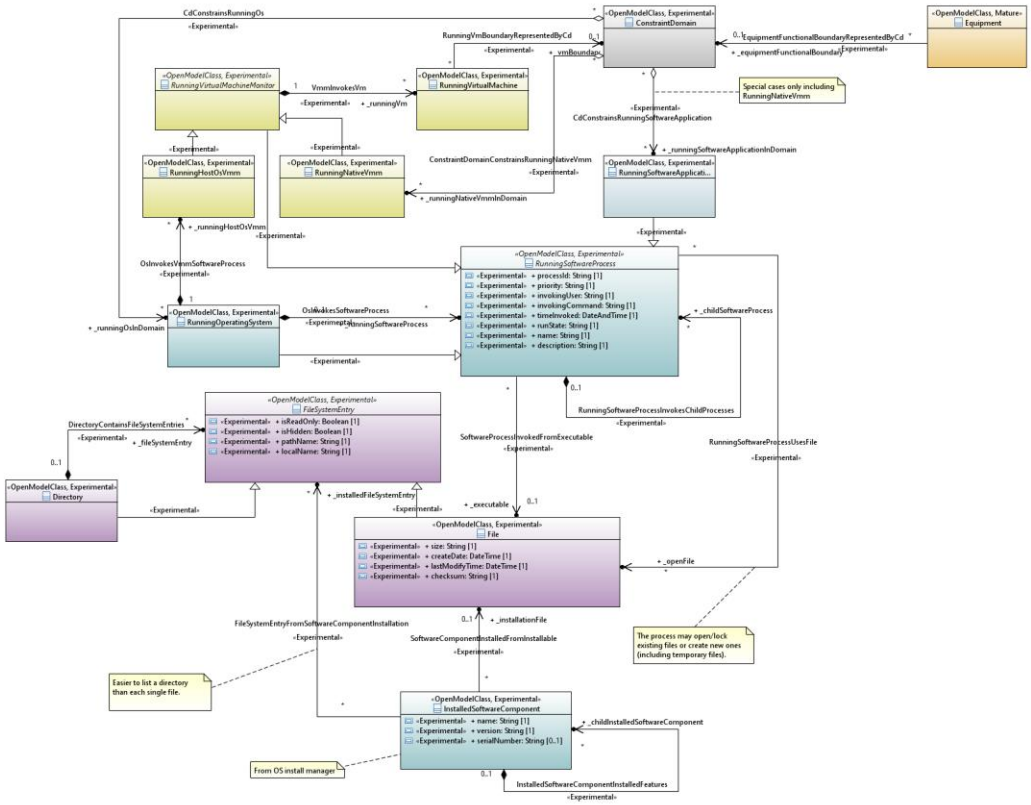
- Native (bare metal) – that run directly on the hardware (type-1)
- Hosted – that run in a host operating system (type-2)

A VMM is used to import, create, run, suspend, halt and delete VMs.

Note that we talk of a VMM running a VM, but whether the VMM runs each VM as a software process or not is not visible to us, so it is not shown as a subclass of RunningSoftwareProcess.

A hosted hypervisor process will be visible to the host operating system, but not the VMs themselves.

RunningVirtualMachine has a ConstraintDomain associated with it (similar to the ConstraintDomain for the physical case). The guest RunningOperatingSystem instances are linked to this ConstraintDomain instance, coupling the RunningVirtualMachine to the guest operating system. Note the linking via ConstraintDomain rather than with direct associations, as this gives the flexibility to mix and match various hardware and software combinations.




CoreModel diagram: Software-VirtualMachine

**Figure 3-3 Skeleton Class Diagram of key object classes**

#### 3.2.3.1 RunningHostOsVmm

Qualified Name: CoreModel::CoreSoftwareModel::VirtualMachine::RunningHostOsVmm

A Virtual Machine Monitor (VMM or Hypervisor) running in a host operating system (type-2).

Inherits properties from:

- RunningVirtualMachineMonitor

This class is Experimental.

#### 3.2.3.2 RunningNativeVmm

Qualified Name: CoreModel::CoreSoftwareModel::VirtualMachine::RunningNativeVmm

A Virtual Machine Monitor (VMM or Hypervisor) running directly on the hardware (bare metal or type-1).

Inherits properties from:

- RunningVirtualMachineMonitor

This class is Experimental.

#### 3.2.3.3 RunningVirtualMachine

Qualified Name: CoreModel::CoreSoftwareModel::VirtualMachine::RunningVirtualMachine

This represents a VirtualMachine that is running, and hence can provide and consume resources. It isn't shown as a subclass of RunningSoftwareProcess as it may not be a running software process and the hypervisor may not allow access to any process related information.
This class is Experimental.

Table 9: Attributes for RunningVirtualMachine

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _vmBoundary | Experimental | Similar to a physical device boundary, we allow a virtual machine to have a constraint boundary. |

#### 3.2.3.4 RunningVirtualMachineMonitor

Qualified Name:
CoreModel::CoreSoftwareModel::VirtualMachine::RunningVirtualMachineMonitor

This is the abstraction of the two different types of VMM.
This class is abstract.

Inherits properties from:

- RunningSoftwareProcess

This class is Experimental.

Table 10: Attributes for RunningVirtualMachineMonitor

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _runningVm | Experimental | The VMs actively running under this VMM. |

### 3.2.4 Container Model

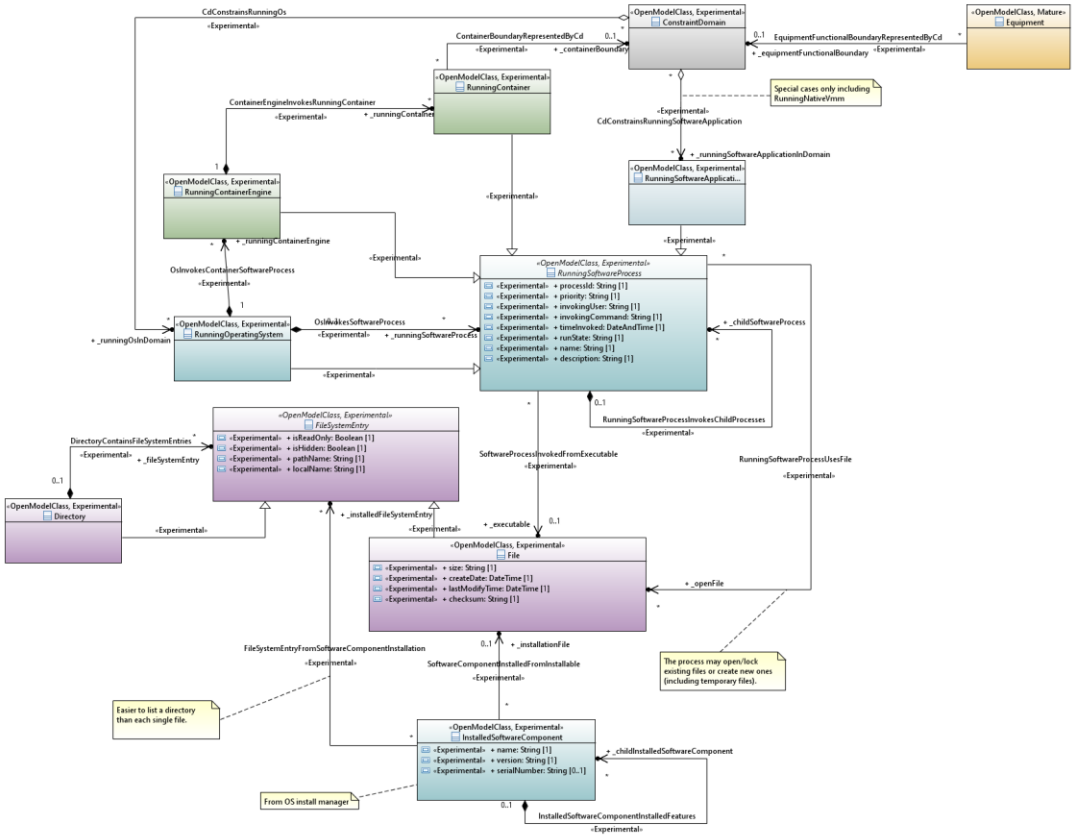
Container (operating system level virtualization) support is very similar to the VMM/ VM case.

A container engine can be thought of as similar to a hosted VMM. Note that there is no bare metal equivalent, a container engine always needs to run in an operating system.

Unlike a VMM, there is no separate guest operating system. The actual details will depend on the container implementation, but it seems reasonable to allow for a software process for the container engine, for each container as well as the application processes[1].

[1] https://unix.stackexchange.com/questions/216618/what-do-the-processes-inside-a-docker-container-look-like

Because a container is more lightweight than a hypervisor, they are often deployed in larger numbers of a smaller size (break an application up into separate containers rather than grouping applications in a VM).




CoreModel diagram: Software-Container

**Figure 3-4 Skeleton Class Diagram of key object classes**

#### 3.2.4.1 RunningContainer

Qualified Name: CoreModel::CoreSoftwareModel::SoftwareContainer::RunningContainer

A container that has been activated by its container engine and hence can provide and consume resources.

Inherits properties from:

- RunningSoftwareProcess

This class is Experimental.

Table 11: Attributes for RunningContainer

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _containerBoundary | Experimental | The constraints defining the boundary of the Container. |

#### 3.2.4.2 RunningContainerEngine

Qualified Name:
CoreModel::CoreSoftwareModel::SoftwareContainer::RunningContainerEngine

A software process that abstracts running applications from the operating system. It provides some level of isolation, but not as much as a hypervisor.

Inherits properties from:

- RunningSoftwareProcess

This class is Experimental.

Table 12: Attributes for RunningContainerEngine

| Attribute Name | Lifecycle Stereotype (empty = Mature) | Description |
|---|---|---|
| _runningContainer | Experimental | The containers running in this container engine. |

### 3.2.5 Software Model in Context

The next step is to make sure the software model is properly integrated into the ONF CIM core model.

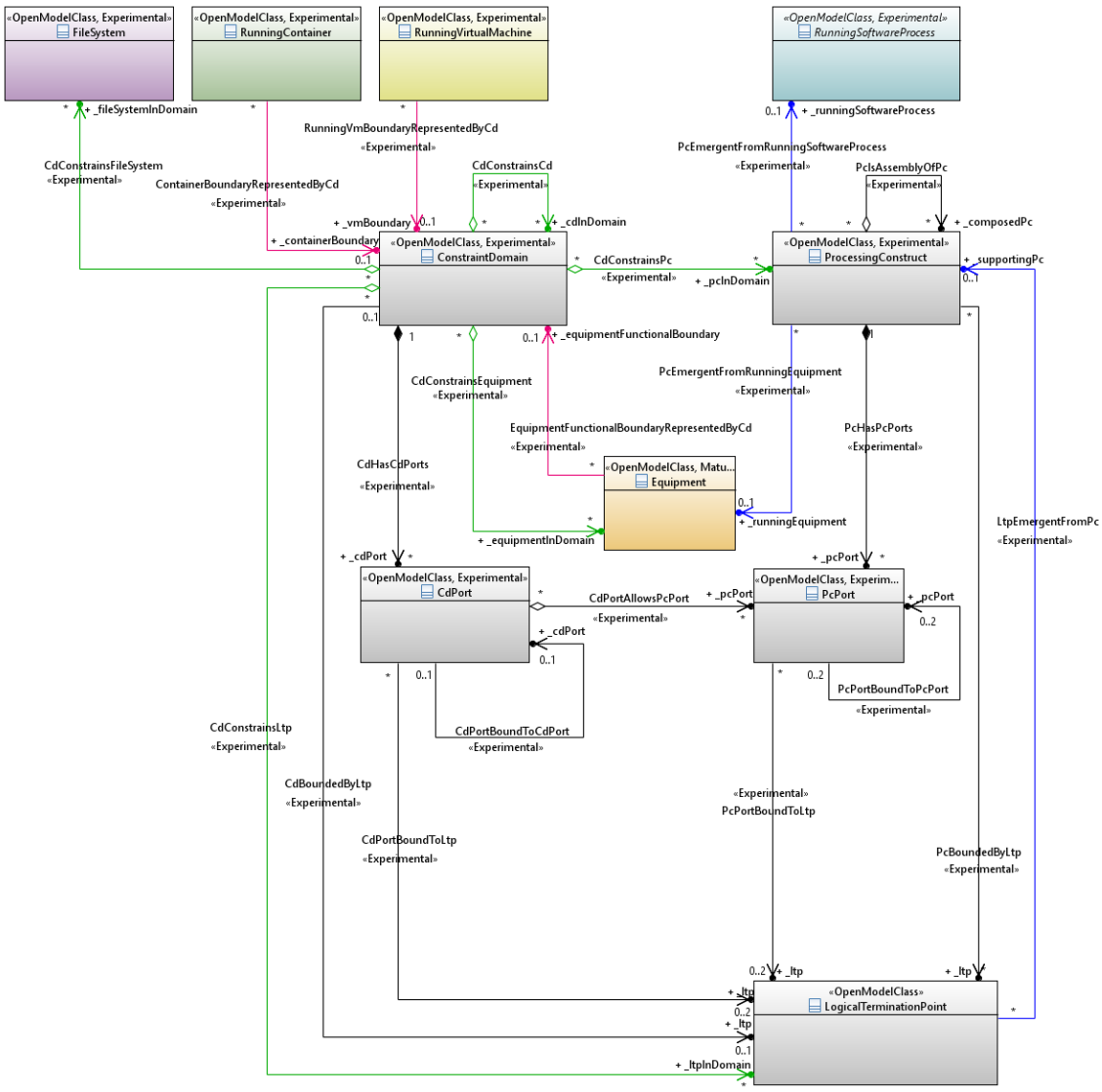
In previous sections in this document, the ConstraintDomain has been used to define the boundary of RunningContainer and RunningVirualMachine in the same way as it is used for Equipment (see the red associations in the figure below).

The ConstraintDomain is also used to provide any other constraints to relevant software entities as it is for other entities in the model (see green associations in the figure below).

Because ConstraintDomain is used as a decoupling class (similar to how LTP is used to decouple the transport part of the model), then only one additional association needs to be added to the software model.

The PcEmergentFromRunningSoftwareProcess association allows functionality to be directly related to a software process (see the relevant blue association in the figure below).

There is an equivalent association to Equipment to represent the emergence of behavior from the physical units. There is also an association LtpEmergentFromPc that shows that the transport functions of a device are provided by non-transport functions modelled as PCs.




CoreModel diagram: Software-SoftwareWithPcAndCd

**Figure 3-5 Software Model in context**

# 4 Software Examples

Examples are provided in TR-512.A.13.

**End of Document**