Core Information Model (CoreModel)

TR-512.15

Compute

Version 1.6
January 2024

ONF Document Type: Technical Recommendation
ONF Document Name: Core Information Model version 1.6

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
1.6	January 2024	Initial Version

1 Introduction

This document is an addendum to the TR-512_v1.5 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%) or 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 and also 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 Compute

This document describes a general high-level model for compute functionality including processing and storage. The model is considered sufficient to represent the capabilities of the compute functions that may be present in a network device or in a controller of network devices.

For storage the document covers management of Block, File and Object storage, both directly attached and over a network. It includes standalone hosts with local storage, Redundant Array of Inexpensive Discs (RAID), Small Computer System Interface (SCSI) as well as network-based storage, including enterprise and cloud storage.

Note that this model excludes physical devices such as Central Processing Unit (CPU) chips and memory chips. All physical device considerations are covered by the existing Equipment model (see TR-512.6).

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

2.1 Background

The compute model covers representation of the functions of CPU, memory and storage.

This model is designed to represent compute architectures in a technology independent manner and is focused on the management and control of the compute functions.

2.1.1 CPU

The original CPU consisted of many physical units, as technology evolved it became possible to implement a CPU in single chip. Now a single chip may contain many CPU cores, and each core may run more than one thread. So, a definition of a 'logical CPU' function is required.

Some chips have a mix of architectures and/or capabilities (asymmetric), others simply have several replications of the same architecture (symmetric).

For example, an asymmetric CPU may have 4 + 4 cores (4 * 1.8 GHz Type-A + 4 * 1.4 GHz Type-B).

The CPU hardware may be housed in an Field Replaceable Unit (FRU) or a non-FRU. This is covered in the Equipment model (see TR-512.6).

2.1.2 Memory

Memory chip(s), Single In-line Memory Module (SIMM) and Dual In-line Memory Modules (DIMM) may be an FRU or non-FRU. This is covered in the Equipment model (see TR-512.6). The model in this document describes the memory functionality (capabilities and capacity).

2.1.3 Storage

2.1.3.1 Challenges

The challenges in producing an abstract, standard storage model include :

- The large number of variations in storage options

- the lack of standard terminology.

For example, the definition of a LUN is problematic :

- "A LUN, is a number used to identify a logical unit, which is a device addressed by the SCSI protocol or Storage Area Network protocols which encapsulate SCSI, such as Fibre Channel or iSCSI" https://en.wikipedia.org/wiki/Logical_unit_number
- "a logical unit *number* (LUN) is a slice or portion of a configured set of disks that is presentable to a host and mounted as a volume within the OS." https://www.computerweekly.com/answer/What-is-a-LUN-and-why-do-we-need-storage-LUNs

2.1.3.2 Storage Options

Storage can be provided in many forms, some of the options commonly used today are shown below.

With each of the options, there could be more than one protocol used, and the diagram shows some of these in pink.

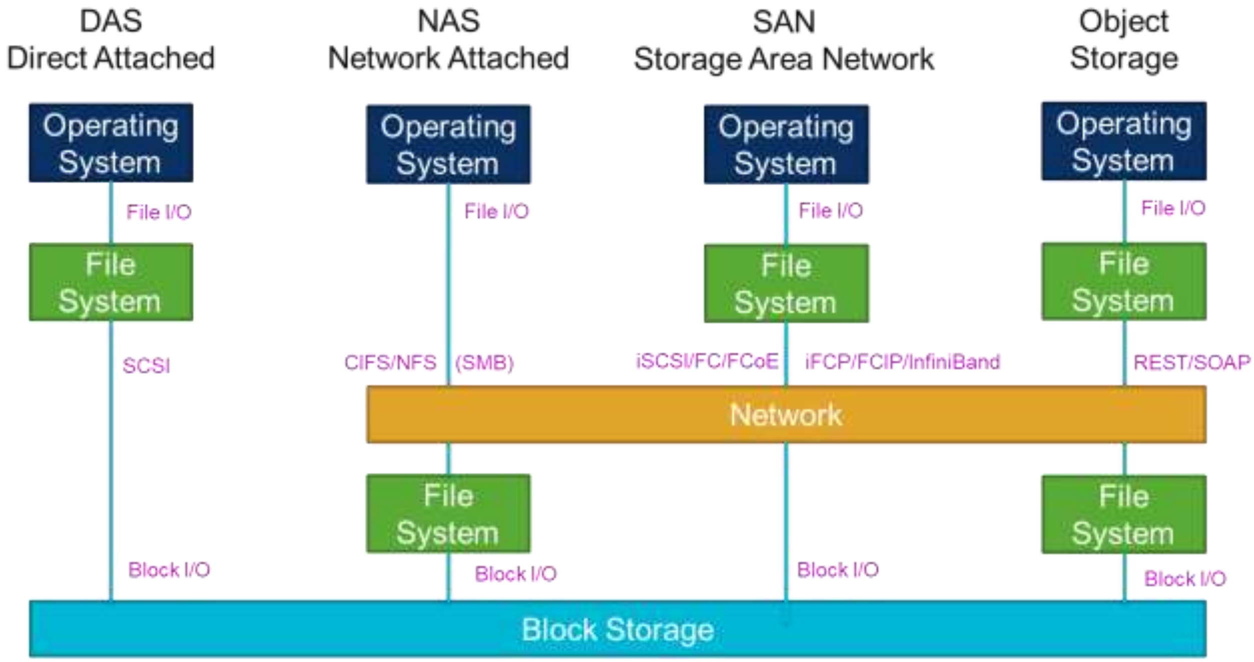


Figure 2-1 – Storage Options

2.1.3.3 Data "at rest"

Both storage and memory allow data to be 'at rest', ready for later retrieval. Memory can perform as volatile storage "RAM drives" or "RAM disks" and storage devices can be used as "virtual memory" where volatile memory is paged in and out of disk to increase the apparent amount of main memory.

The storage model is used for both storage and memory. It covers both the case where the access is to files, blocks or objects using a storage protocol and the case where access is to locations via the memory protocols.

2.2 Storage Extent

In the model set out in this document, StorageExtent is defined as the key unit of storage capacity that the rest of the model is built around. The extent is a block or segment of storage (contiguous bytes).

The model covers ranges of extents.

2.3 Partitioning and Aggregation

The model supports both partitioning and aggregation of StorageExtent ranges.

The simple example below, where StorageExtent range is represented as a 'piece of tape', should help clarify the concepts.

Figure 2-2 - StorageExtent as a 'piece of tape'

The tape can be aggregated in two ways, by end-to-end concatenation and by striping.

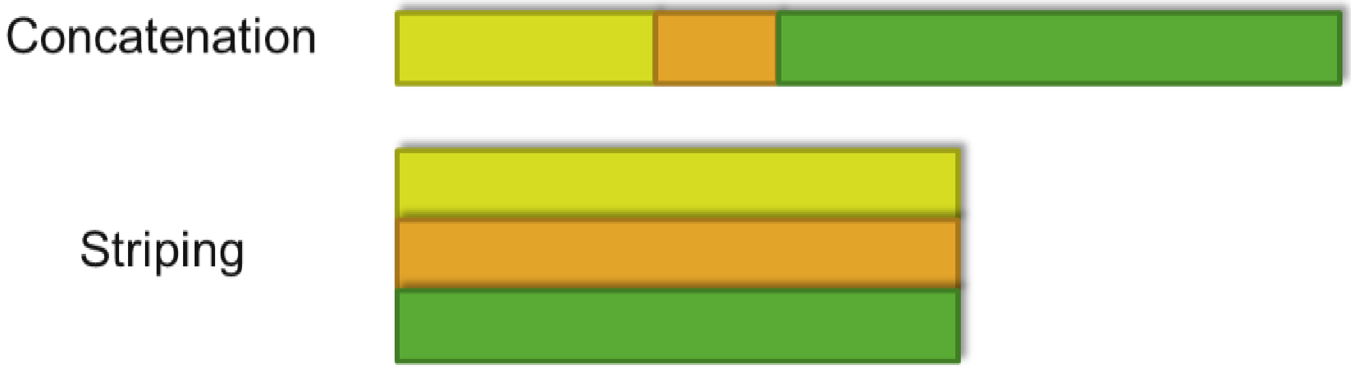


Figure 2-3 – StorageExtent concatenation and striping

The tape can be cut into sub extents, i.e., can be partitioned (the opposite of concatenation).

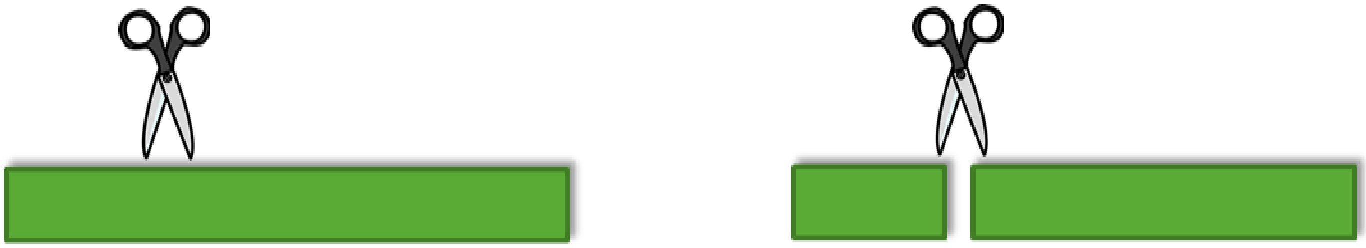


Figure 2-4 – StorageExtent partitioning

Note that the operations are 'closed' that is both the inputs and the results of the operations are StorageExtent ranges, allowing the operations to be performed recursively.

2.4 Storage Pooling

Originally, due to the limitations of the hardware, only local storage was available. As technology evolved, the support of shared storage (provided over a network) became feasible. Therefore, the model needs to support pooling of physical storage that can then be allocated logically to various consumers.

To do that, the model defines a ComputePool with ComputePoolEntries.

Note that the decision was made to have a single compute pool rather than separate Storage, CPU and memory pools[1], because :

- CPU and memory are usually tightly coupled, and the pool can then allocate these consistently
- *Sometimes* storage is tightly coupled with CPU and memory and the pool can then allocate these consistently

There are two types of entries :

- Pool inputs for SSD, PhysicalDisk, VM VirtualDisk and LogicalEntry
- A pool output as an extent allocation (volume)

Note that the extent allocation from a pool can be a LogicalEntry to another pool allowing for allocation chaining.

Note also that the pools aren't hierarchical (deliberately no ComputePool contained in self-join)

- The association StorageExtentPoolEntryIsLogical allows an output from one pool to become the input of another pool
- This needs to form a directed acyclic graph (no loops)

Note that there is no association linking the pool inputs and outputs. The ordering of the inputs allows the input to output extent mapping to be determined.

It is assumed that there will be many simple pools rather than few large complex pools with complex mappings.

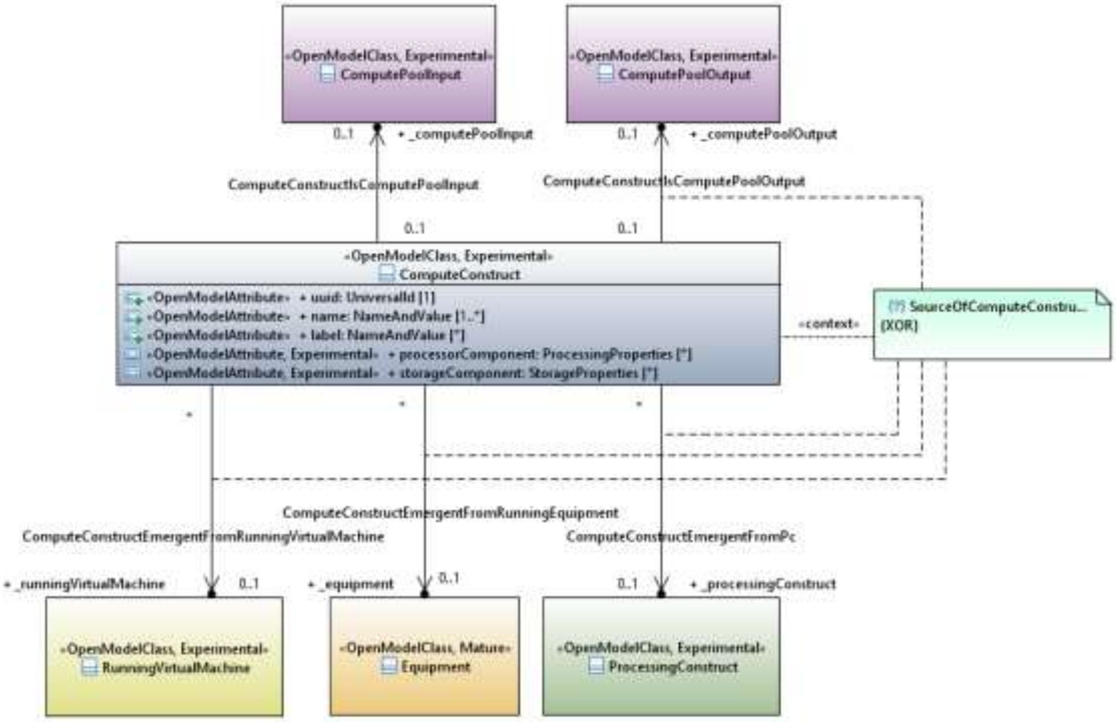
---

[1] Note that this document only considers storage entries.

3 Compute model and context

3.1 ComputeConstruct positioning

The following figures set out the core of the compute model.


«OpenModelClass, Experimental»
ComputePoolInput
«OpenModelClass, Experimental»
ComputePoolOutput
0..1
+ _computePoolInput
0..1
+ _computePoolOutput
ComputeConstructIsComputePoolInput
ComputeConstructIsComputePoolOutput
0..1
0..1
«OpenModelClass, Experimental»
ComputeConstruct
«OpenModelAttribute» + uuid: UniversalId [1]
«OpenModelAttribute» + name: NameAndValue [1..*]
«OpenModelAttribute» + label: NameAndValue [*]
«OpenModelAttribute, Experimental» + processorComponent: ProcessingProperties [*]
«OpenModelAttribute, Experimental» + storageComponent: StorageProperties [*]
«context»
{?} SourceOfComputeConstru...
{XOR}
*
*
*
ComputeConstructEmergentFromRunningEquipment
ComputeConstructEmergentFromRunningVirtualMachine
ComputeConstructEmergentFromPc
+ _runningVirtualMachine
0..1
+ _equipment
0..1
0..1
+ _processingConstruct
«OpenModelClass, Experimental»
RunningVirtualMachine
«OpenModelClass, Mature»
Equipment
«OpenModelClass, Experimental»
ProcessingConstruct

CoreModel diagram: Compute-ComputeStructure

Figure 3-1 Compute Structure

3.1.1 ComputeConstruct

Qualified Name: Compute::ComputeConstruct::ComputeConstruct

An assembly of processing and storage functionality that provides an overall compute capability.
The interconnect of the processing and storage is not detailed. The purpose of this model is to deal primarily with resource allocation and usage.
At this stage this is a degenerate form of component-system pattern.
It is expected that the compute construct will gain ports in a later development and that some detailed modeling of the system may be necessary.

Inherits properties from:

- GlobalClass

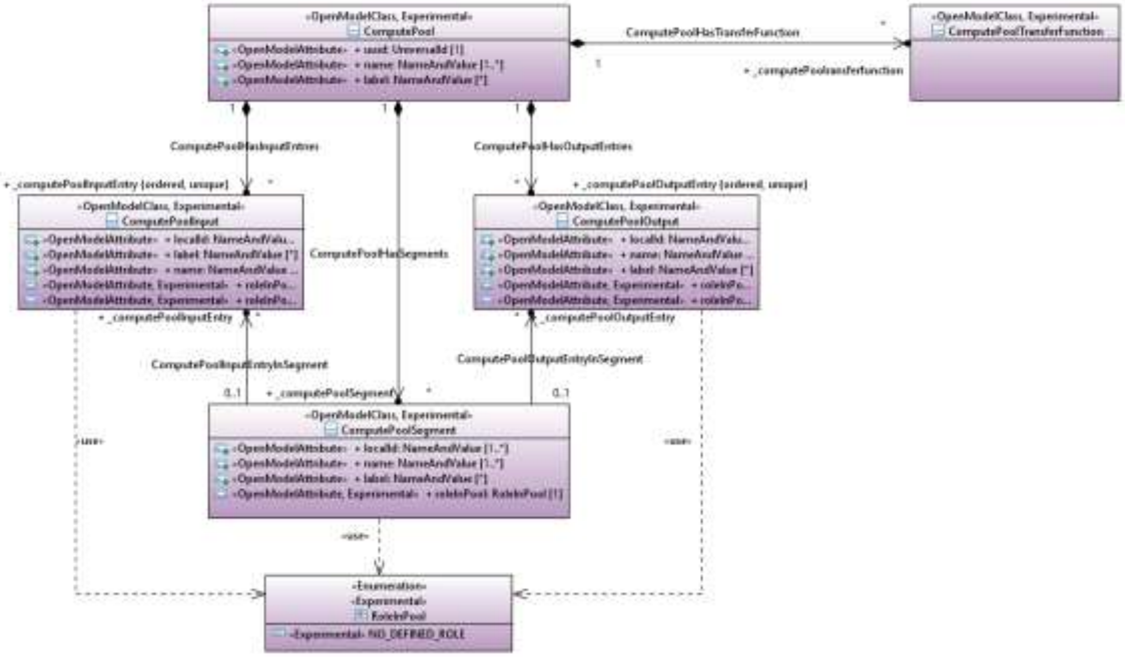
This class is Experimental.

Table 1: Attributes for ComputeConstruct

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
processorComponent	Experimental	The properties of the processing capability of a compute construct. The compute construct may have several distinct processing forms.
storageComponent	Experimental	The properties of the storage capability of a compute construct. The compute construct may have several distinct storage forms. Note that storage covers any element that has the purposeful capability of retaining data (e.g., RAM, a buffer, a queue).
_processingConstruct	Experimental	The processing construct that gives rise to the compute construct.
_equipment	Experimental	The equipment that gives rise to the compute construct.
_computePoolInput	Experimental	The usage of the compute construct in a pool.
_computePoolOutput	Experimental	The compute pool output that gives rise to the compute construct.
_runningVirtualMachine	Experimental	The virtual machine that gives rise to the compute construct.
_fileSystem	Experimental	The file system that gives rise to the compute construct.
_runningSoftwareProcess	Experimental	The software process that gives rise to the compute construct.

3.2 ComputePool

This part of the model allows for basic allocation of compute resources.


«OpenModelClass, Experimental»
ComputePool
«OpenModelAttribute» + uuid: UniversalId [1]
«OpenModelAttribute» + name: NameAndValue [1..*]
«OpenModelAttribute» + label: NameAndValue [*]
ComputePoolHasTransferFunction
1
*
+ _computePooltransferfunction
«OpenModelClass, Experimental»
ComputePoolTransferFunction
1
1
1
ComputePoolHasInputEntries
ComputePoolHasOutputEntries
+ _computePoolInputEntry {ordered, unique}
*
*
+ _computePoolOutputEntry {ordered, unique}
«OpenModelClass, Experimental»
ComputePoolInput
«OpenModelAttribute» + localId: NameAndValu...
«OpenModelAttribute» + label: NameAndValue [*]
«OpenModelAttribute» + name: NameAndValue ...
«OpenModelAttribute, Experimental» + roleInPo...
«OpenModelAttribute, Experimental» + roleInPo...
ComputePoolHasSegments
«OpenModelClass, Experimental»
ComputePoolOutput
«OpenModelAttribute» + localId: NameAndValu...
«OpenModelAttribute» + name: NameAndValue ...
«OpenModelAttribute» + label: NameAndValue [*]
«OpenModelAttribute, Experimental» + roleInPo...
«OpenModelAttribute, Experimental» + roleInPo...
+ _computePoolInputEntry
*
*
+ _computePoolOutputEntry
ComputePoolInputEntryInSegment
ComputePoolOutputEntryInSegment
0..1
+ _computePoolSegment
*
0..1
«use»
«OpenModelClass, Experimental»
ComputePoolSegment
«OpenModelAttribute» + localId: NameAndValue [1..*]
«OpenModelAttribute» + name: NameAndValue [1..*]
«OpenModelAttribute» + label: NameAndValue [*]
«OpenModelAttribute, Experimental» + roleInPool: RoleInPool [1]
«use»
«use»
«Enumeration»
«Experimental»
RoleInPool
«Experimental» NO_DEFINED_ROLE

CoreModel diagram: Compute-ComputePoolDetail

Figure 3-2 Compute Pool

3.2.1 ComputePool

Qualified Name: Compute::ComputePool::ComputePool

A pool that manages ComputePoolEntries. These entries can consist of Storage, Memory and CPU resources.
This mechanism allows for basic allocation of groups of compute resource.
The pool does allow for some combination of resources but is not modelled as a system of components, it is simply a basic container.

Inherits properties from:

- GlobalClass

This class is Experimental.

Table 2: Attributes for ComputePool

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
_computePoolInputEntry	Experimental	The resources provided to the pool.
_computePoolOutputEntry	Experimental	The resources exposed by the pool.
_computePoolSegment	Experimental	Pool segments (grouping and structures).

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
_computePoolransferfunction	Experimental	Explains the mapping from the inputs to the pool to the outputs from the pool, i.e., how the output is produced.

3.2.2 ComputePoolInput

Qualified Name: Compute::ComputePool::ComputePoolInput

A compute capability provided to the pool.

Inherits properties from:

- LocalClass

This class is Experimental.

Table 3: Attributes for ComputePoolInput

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
roleInPool	Experimental	Explains the use of the item in the pool and especially with respect to the mapping.
roleInPoolSegment	Experimental	Role (perhaps just position) of the pool item in the related segment. Provides information on the concatenation process (of either inputs or outputs).

3.2.3 ComputePoolOutput

Qualified Name: Compute::ComputePool::ComputePoolOutput

A compute capability derived from the pool.

Inherits properties from:

- LocalClass

This class is Experimental.

Table 4: Attributes for ComputePoolOutput

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
roleInPool	Experimental	Explains the use of the item in the pool and especially with respect to the mapping.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
roleInPoolSegment	Experimental	Role (perhaps just position) of the pool item in the related segment. Provides information on the concatenation process (of either inputs or outputs).

3.2.4 ComputePoolSegment

Qualified Name: Compute::ComputePool::ComputePoolSegment

A grouping of inputs and/or outputs.
Used where several inputs and/or outputs need to be considered as a unit.

Inherits properties from:

- LocalClass

This class is Experimental.

Table 5: Attributes for ComputePoolSegment

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
roleInPool	Experimental	Explains the use of the item in the pool and especially with respect to the mapping.
_computePoolInputEntry	Experimental	Collection of inputs where each may have a role stated for the segment.
_computePoolOutputEntry	Experimental	Collection of outputss where each may have a role stated for the segment.

3.2.5 ComputePoolTransferFunction

Qualified Name: Compute::ComputePool::ComputePoolTransferFunction

Placeholder.

Explains the mapping between inputs and outputs of the pool.

This class is Experimental.

3.2.6 RoleInPool

Qualified Name: Compute::ComputePool::RoleInPool

The role of the item in the pool.

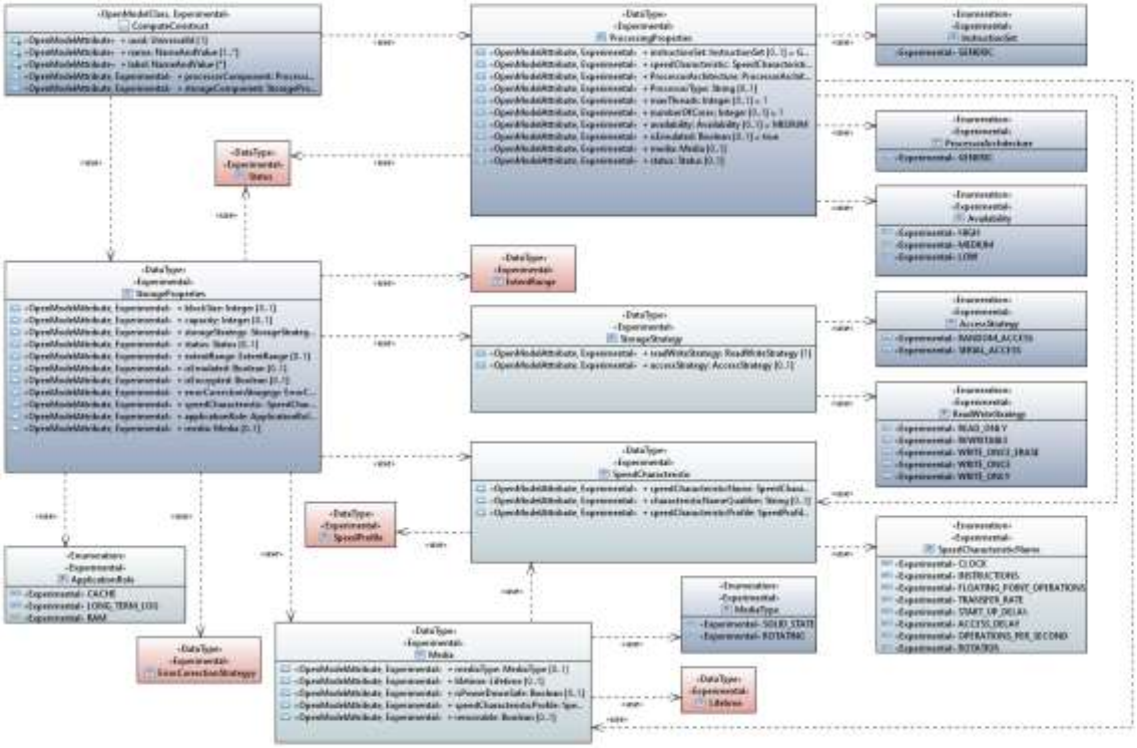
Applied stereotypes:

- Experimental

Contains Enumeration Literals:

- NO_DEFINED_ROLE:
    - The item has no defined role, the pool is simply grouping several items.
    - Applied stereotypes:
        - Experimental

3.3 Compute model data types



CoreModel diagram: Compute-DataTypes

Figure 3-3 Compute Data Tyoes

3.3.1 ProcessingProperties

Qualified Name: Compute::Processing::ProcessingProperties

The properties of the processing capability of a compute construct.

Table 6: Attributes for ProcessingProperties

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
instructionSet	Experimental	The identifier of the set of instructions that can be used to manipulate the processing functionality.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
speedCharacteristic	Experimental	A statement of operational performance. Note that this is a somewhat simplistic property. Speed/performance is far more complex and requires a temporal treatment.
ProcessorArchitecture	Experimental	The identification of the architecture of the processing functionality. Essentially a reference to a spec.
ProcessorType	Experimental	The identification of the specification of the processor. Note that the processor may be a hardware device or a software emulation.
maxThreads	Experimental	Manximum number of threads. Note that this is a somewhat simplistic property. It requires a temporal treatment and may have other dependencies.
numberOfCores	Experimental	Number of cores.
availability	Experimental	A statement of the operational resilience of the function.
isEmulated	Experimental	An indication as to whether the function is a native form of the type or is emergent from a complex system designed to appear to be the type.
media	Experimental	Details of the physical realization. Note that this should migrate to the physical model.
status	Experimental	A statement of the operational situation of the function.

3.3.2 StorageProperties

Qualified Name: Compute::Storage::StorageProperties

The properties of the storage capability of a compute construct.

Table 7: Attributes for StorageProperties

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
blockSize	Experimental	Where the storage is arranged in blocks, this property provides the size of the blocks.
capacity	Experimental	Size of the storage in total.
storageStrategy	Experimental	Defines how the data is stored and accessed.
status	Experimental	A statement of the operational situation of the function.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
extentRange	Experimental	Represents the range of the extent, i.e., the storage capacity.
isEmulated	Experimental	An indication as to whether the function is a native form of the type or is emergent from a complex system designed to appear to be the type.
isEncrypted	Experimental	The data is encrypted. The encryption scheme is not described here. Further work is required to detail this area.
errorCorrectionStragegy	Experimental	Some mechanism is used to correct errors in the data.
speedCharacteristic	Experimental	A statement of operational performance. Note that this is a somewhat simplistic property. Speed/performance is far more complex and requires a temporal treatment.
applicationRole	Experimental	Indicates the design intent for the application of this storage function.
media	Experimental	Details of the physical realization. Note that this should migrate to the physical model.

3.3.3 StorageStrategy

Qualified Name: Compute::Storage::StorageStrategy

Defines how the data is stored and accessed.

Table 8: Attributes for StorageStrategy

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
readWriteStrategy	Experimental	Approach to examine and change of the data stored.
accessStrategy	Experimental	Approach to access of the data.

3.3.4 ErrorCorrectionStrategy

Qualified Name: Compute::Storage::ErrorCorrectionStrategyy

Placeholder
Some mechanism is used to correct errors in the data.
The specific mechanism have not yet been modelled.

3.3.5 ExtentRange

Qualified Name: Compute::Storage::ExtentRange

Placeholder.
Represents the range of the extent, i.e., the storage capacity.
Further work is required to develop the form of representation.

3.3.6 Media

Qualified Name: Compute::Media::Media

Details of the physical realization.
Note that this should migrate to the physical model.

Table 9: Attributes for Media

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
mediaType	Experimental	Type of the media.
lifetime	Experimental	Specification of characteristics of expected operating lifetime.
isPowerDownSafe	Experimental	Can be powered down without losing or corrupting state.
speedCharacteristicProfile	Experimental	A statement of operational performance. Includes ROTATION speed for disks. Note that this is a somewhat simplistic property. Speed/performance is far more complex and requires a temporal treatment.
removable	Experimental	Can be removed during operation.

3.3.7 Lifetime

Qualified Name: Compute::Media::Lifetime

Placeholder.
Specification of characteristics of expected operating lifetime.

3.3.8 SpeedCharacteristic

Qualified Name: Compute::Common::SpeedCharacteristic

A statement of operational performance.
Note that this is a somewhat simplistic property. Speed/performance is far more complex and requires a temporal treatment.

Table 10: Attributes for SpeedCharacteristic

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
speedCharacteristicName	Experimental	Name of the speed characteristic type.

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
speedCharacteristicProfile	Experimental	Defines the complex speed characteristics.
characteristicNameQualifier	Experimental	Additional qualification of the characteristic name (e.g., when there are two clocks, both will be CLOCK and need to be distinguished).

3.3.9 Status

Qualified Name: Compute::Common::Status

Placeholder.
A statement of the operational situation of the function.

3.3.10 AccessStrategy

Qualified Name: Compute::Storage::AccessStrategy

List of access approaches.

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- RANDOM_ACCESS:
  - Any individual item of data can be accessed directly with equal performance.
  - Applied stereotypes:
    - Experimental
- SERIAL_ACCESS:
  - Data can be accessed efficiently in series, but accessing the first element of data takes significant time.
  - Applied stereotypes:
    - Experimental

3.3.11 ReadWriteStrategy

Qualified Name: Compute::Storage::ReadWriteStrategy

List of read and write combinations.

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- READ_ONLY:
  - The data can be read but not written.
  - Applied stereotypes:

        - Experimental
- REWRITABLE:
    - The data can be read and written as required.
    - Applied stereotypes:
        - Experimental
- WRITE_ONCE_ERASE:
    - The data can be read as required and can be written one unit at a time but cannot be changed.
      The data can only be errased on mass (all data at once).
    - Applied stereotypes:
        - Experimental
- WRITE_ONCE:
    - Data can be read as required but written only once and never changed or erased.
    - Applied stereotypes:
        - Experimental
- WRITE_ONLY:
    - Data can be written but not read.
    - Applied stereotypes:
        - Experimental

3.3.12 ApplicationRole

Qualified Name: Compute::Storage::ApplicationRole

List of application roles for a storage function.

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- CACHE:
    - Temporary storage for frequently accessed data.
    - Applied stereotypes:
        - Experimental
- LONG_TERM_LOG:
    - For storage of large quantities of data where there are essentially no changes, just additions and the data needs to be stored for a very long duration.
    - Applied stereotypes:
        - Experimental
- RAM:
    - Main memory.
    - Applied stereotypes:
        - Experimental

3.3.13 SpeedProfile

Qualified Name: Compute::Common::SpeedProfile

Placeholder.
Defines the complex speed characteristics.

Applied stereotypes:

- Experimental

3.3.14 SpeedCharacteristicName

Qualified Name: Compute::Common::SpeedCharacteristicName

List of speed characteristics.
Note that each could have mean, max, min etc.

Applied stereotypes:

- Experimental

Contains Enumeration Literals:

- CLOCK:
  - Clock ticks.
  - Applied stereotypes:
    - Experimental
- INSTRUCTIONS:
  - Instructions in the native language of the processor.
  - Applied stereotypes:
    - Experimental
- FLOATING_POINT_OPERATIONS:
  - Floating point operations.
  - Applied stereotypes:
    - Experimental
- TRANSFER_RATE:
  - Data transfer into/out of the function.
  - Applied stereotypes:
    - Experimental
- START_UP_DELAY:
  - Time from initialization to full operation.
  - Applied stereotypes:
    - Experimental
- ACCESS_DELAY:
  - Delay in gaining access to the function.
  - Applied stereotypes:
    - Experimental
- OPERATIONS_PER_SECOND:
  - Generalized activities per second.
  - Applied stereotypes:
    - Experimental
- ROTATION:
  - Rotation speed of a mechanical device.

    - Applied stereotypes:
        - Experimental

3.3.15 InstructionSet

Qualified Name: Compute::Processing::InstructionSet

List of the identifiers of the set of instruction sets.
Essentially spec references.

Applied stereotypes:

- Experimental

Contains Enumeration Literals:

- GENERIC:
    - Generic instruction set.
    - Applied stereotypes:
        - Experimental

3.3.16 ProcessorArchitecture

Qualified Name: Compute::Processing::ProcessorArchitecture

List of identifiers for architectures of processing functionality.
Essentially spec references.

Applied stereotypes:

- Experimental

Contains Enumeration Literals:

- GENERIC:
    - Generic architecture.
    - Applied stereotypes:
        - Experimental

3.3.17 Availability

Qualified Name: Compute::Processing::Availability

List of statements of the operational resilience of the functions.

Applied stereotypes:

- Experimental

Contains Enumeration Literals:

- HIGH:
    - High operational resilience (nearly always on).
    - Applied stereotypes:
        - Experimental

- MEDIUM:
    - Medium operational resilience (normally running OK).
    - Applied stereotypes:
        - Experimental
- LOW:
    - Low operational resilience (often not functioning).
    - Applied stereotypes:
        - Experimental

3.3.18 MediaType

Qualified Name: Compute::Media::MediaType

List of types of media.

Applied stereotypes:

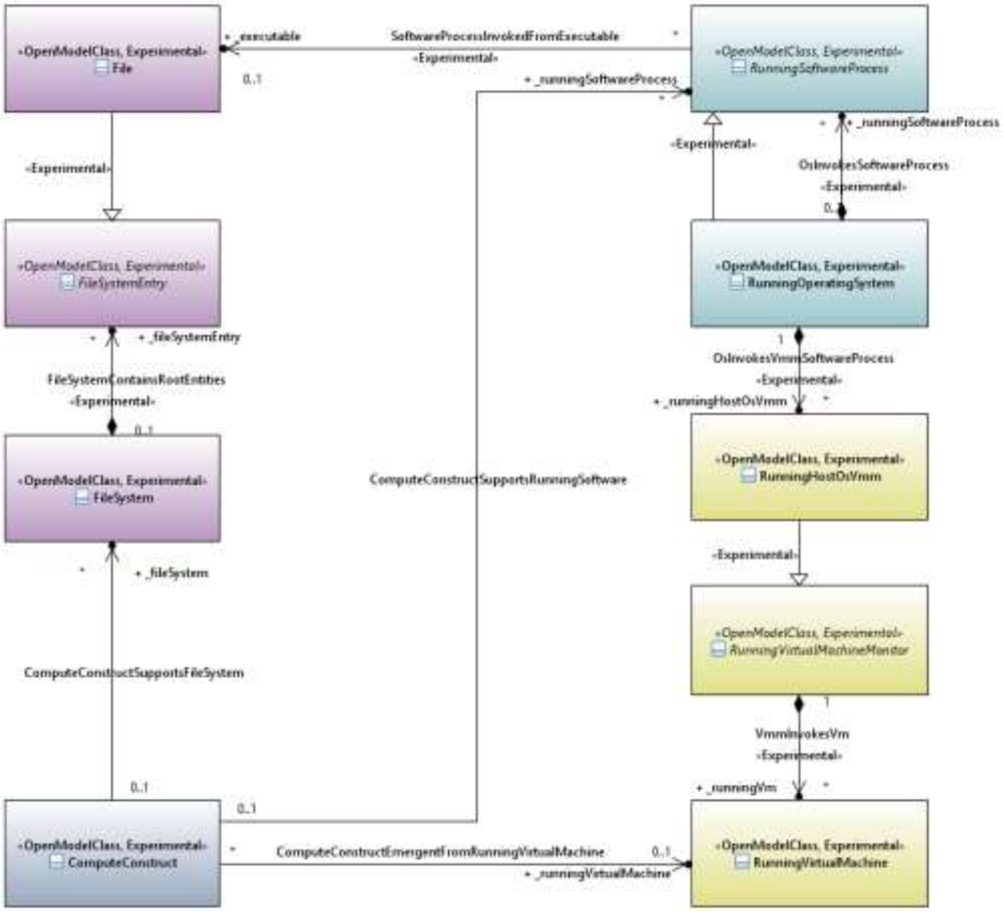
- Experimental

Contains Enumeration Literals:

- SOLID_STATE:
    - Transistor etc. based with no mechanically active parts.
    - Applied stereotypes:
        - Experimental
- ROTATING:
    - Mechanical and rotating.
    - Applied stereotypes:
        - Experimental

3.4 Relationship to File System and Software

The following diagram shows the relationships between this compute model and other existing models, such as the software model described in TR-512.12.


«OpenModelClass, Experimental»
File
+ _executable
0..1
SoftwareProcessInvokedFromExecutable
«Experimental»
*
«OpenModelClass, Experimental»
RunningSoftwareProcess
+ _runningSoftwareProcess
*
«Experimental»
*
+ _runningSoftwareProcess
OsInvokesSoftwareProcess
«Experimental»
0..1
«OpenModelClass, Experimental»
FileSystemEntry
*
+ _fileSystemEntry
FileSystemContainsRootEntities
«Experimental»
0..1
«OpenModelClass, Experimental»
RunningOperatingSystem
1
OsInvokesVmmSoftwareProcess
«Experimental»
+ _runningHostOsVmm
*
«OpenModelClass, Experimental»
FileSystem
ComputeConstructSupportsRunningSoftware
«OpenModelClass, Experimental»
RunningHostOsVmm
*
+ _fileSystem
«Experimental»
«OpenModelClass, Experimental»
RunningVirtualMachineMonitor
ComputeConstructSupportsFileSystem
1
VmmInvokesVm
«Experimental»
+ _runningVm
*
0..1
0..1
«OpenModelClass, Experimental»
ComputeConstruct
*
ComputeConstructEmergentFromRunningVirtualMachine
0..1
+ _runningVirtualMachine
«OpenModelClass, Experimental»
RunningVirtualMachine

CoreModel diagram: Compute-ComputeAndSoftware

Figure 3-4 Compute and Software

4 A simple compute example

This simple example shows how the concepts in the model fit together. The figure below shows an assembly of hardware.

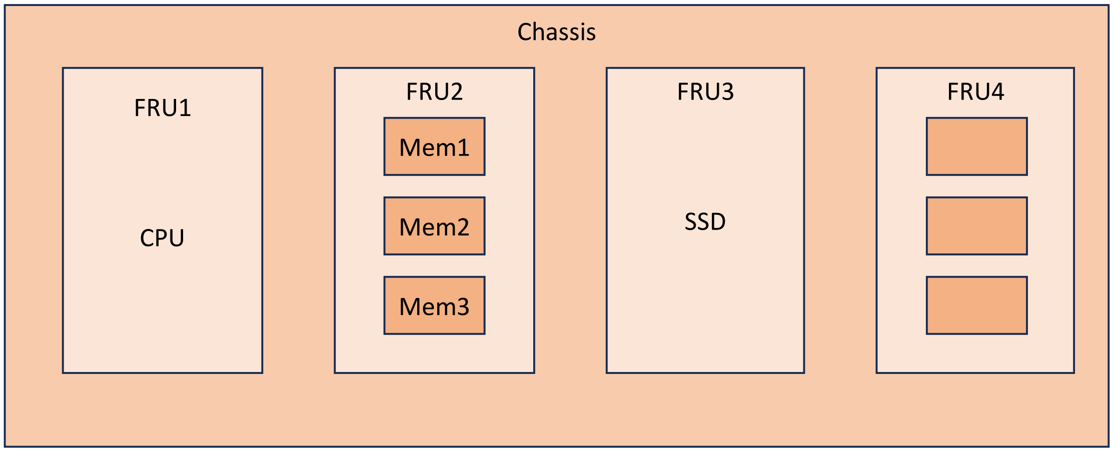
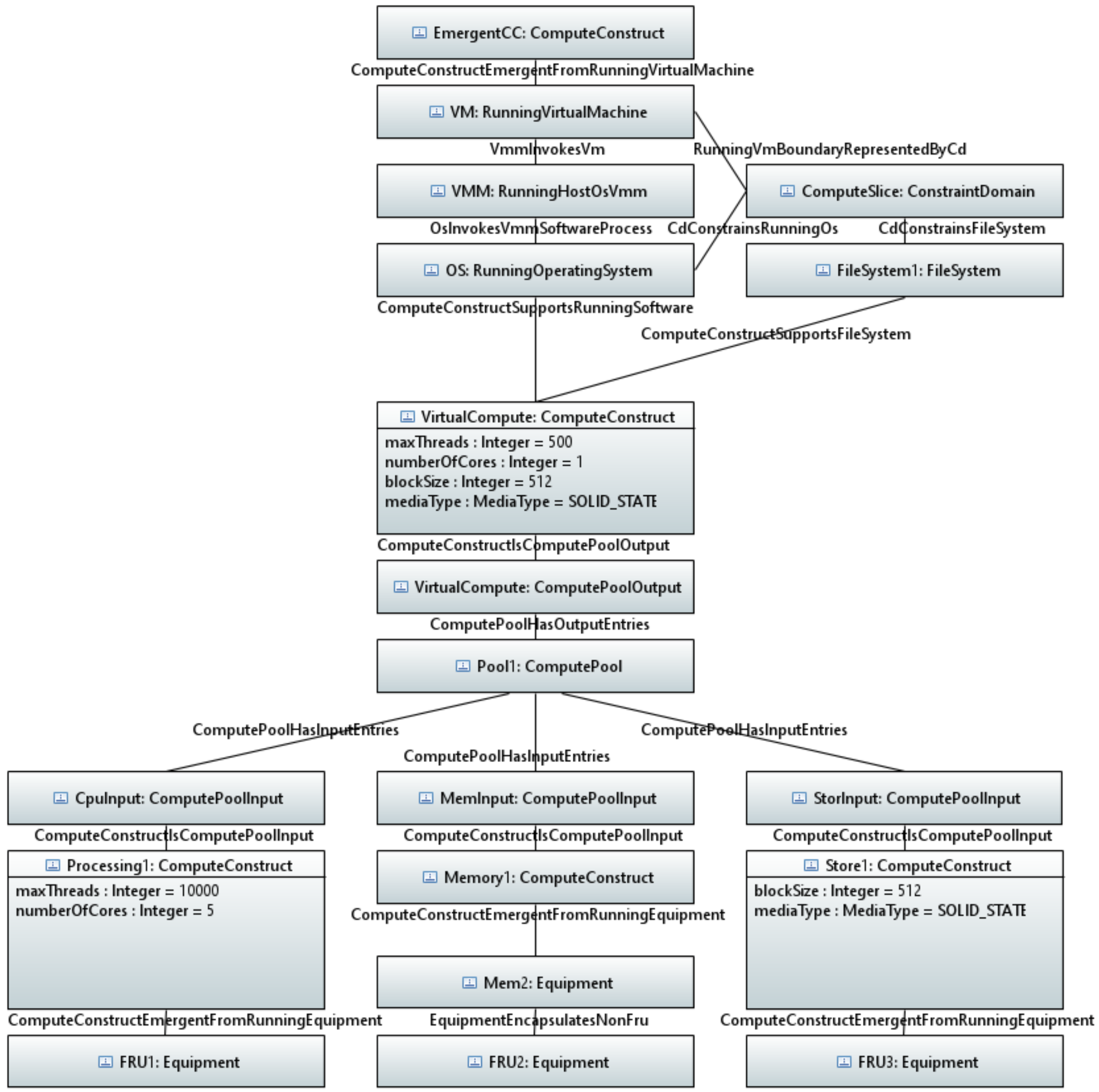


Figure 4-1 - Simple Compute hardware example

The instance representation below shows how a slice of the capability of the hardware can be represented using the compute model. Not all of the hardware is required in the slice depicted and some associations in the software model are omitted.


EmergentCC: ComputeConstruct
ComputeConstructEmergentFromRunningVirtualMachine
VM: RunningVirtualMachine
VmmInvokesVm
RunningVmBoundaryRepresentedByCd
VMM: RunningHostOsVmm
ComputeSlice: ConstraintDomain
OsInvokesVmmSoftwareProcess
CdConstrainsRunningOs
CdConstrainsFileSystem
OS: RunningOperatingSystem
FileSystem1: FileSystem
ComputeConstructSupportsRunningSoftware
ComputeConstructSupportsFileSystem
VirtualCompute: ComputeConstruct
maxThreads : Integer = 500
numberOfCores : Integer = 1
blockSize : Integer = 512
mediaType : MediaType = SOLID_STATE
ComputeConstructIsComputePoolOutput
VirtualCompute: ComputePoolOutput
ComputePoolHasOutputEntries
Pool1: ComputePool
ComputePoolHasInputEntries
ComputePoolHasInputEntries
ComputePoolHasInputEntries
CpuInput: ComputePoolInput
MemInput: ComputePoolInput
StorInput: ComputePoolInput
ComputeConstructIsComputePoolInput
ComputeConstructIsComputePoolInput
ComputeConstructIsComputePoolInput
Processing1: ComputeConstruct
maxThreads : Integer = 10000
numberOfCores : Integer = 5
Memory1: ComputeConstruct
Store1: ComputeConstruct
blockSize : Integer = 512
mediaType : MediaType = SOLID_STATE
ComputeConstructEmergentFromRunningEquipment
Mem2: Equipment
ComputeConstructEmergentFromRunningEquipment
EquipmentEncapsulatesNonFru
ComputeConstructEmergentFromRunningEquipment
FRU1: Equipment
FRU2: Equipment
FRU3: Equipment

CoreModel diagram: Compute-SimpleCompute

Figure 4-2 Compute example instance diagram

5 Model considerations

5.1 Pooling

The decision was made to have a single compute pool rather than separate Storage, CPU and memory pools[2], because :

- CPU and memory are usually tightly coupled (via the CPU and memory buses) and the pool can then allocate these consistently
- *Sometimes* storage is tightly coupled with CPU and memory and the pool can then allocate these consistently (local attached un-sharable storage)

The pools aren't hierarchical, i.e., there is not a self-join association on ComputePool (deliberately), because:

- The association StorageExtentPoolEntryIsLogical allows an output from one pool to become the input of another pool
- The structure needs to form a directed acyclic graph (no loops)

Note that there is no association linking the pool inputs and outputs. The ComputePoolTransferFunction will determine the order and association between inputs and outputs. In addition, the ordering of the inputs allows the input-to-output extent mapping to be determined in simple cases.

It is assumed that there will be a large number of simple pools rather than few large complex pools with complex mappings.

5.2 Partitioning and Aggregation

The partitioning and aggregation of CPU and memory is subtly different from that of Storage.

With Storage, in theory each extent can be considered separately. Some extents could come from a disk A, some from disk B. With a sensible pool allocation there is no need to worry about segmenting the pool.

With CPU and storage, we have 2 issues :

1. The pool needs to remember and enforce segmentation (resource chunks)
2. The pool needs to pair CPU and Memory segments

For example, if there are 2 blade servers Blade-A and Blade-B.

- A software process cannot be allocated a CPU from Blade-A and memory from Blade-B.
- A software thread cannot be allocated half its CPU requirements from Blade-A and half from Blade-B.
- A software thread cannot be allocated half its memory requirements from Blade-A and half from Blade-B.

---

[2] Note that this document only considers CPU and memory entries.

A multi-threaded application may be able to run across multiple CPU/memory pairs, but it would have to know the segmentation.

This model will use the following :

- ComputePoolSegment is defined to allow the definition of 'segments'
- Inputs to the pool can be optionally assigned to a segment
- Each pool input/output may optionally be related to one segment
- If an input entry relates to a segment, then no output can be assigned that 'crosses' the segment (i.e., each output can only relate to 1 segment)
- Multiple inputs can only be combined if both do not relate to a segment or if both relate to the same segment
- If an input entry is related to a segment, then it should be propagated to any outputs that relate to the same segment
- If a CPU output relates to a segment, then it can only be used with memory that relates to the same segment (and visa-versa)
- Each segment should have its own internal number range in the pool

5.3 Items for Further Investigation

5.3.1 ComputePoolTransferFunction

This is currently a placeholder. The transfer function model will be developed in a future release. The model should take advantage of the work on the modeling of Task and ViewMappingFunction both of which have elements of transfer function. The work will also need to leverage the spec model patterns.

The transfer function in this model is intended to be basic. A more sophisticated form could be developed.

5.3.2 Application of specification model

The spec model approach should be applied to all aspects of this model.

Some of the properties of the compute and storage could be moved to spec model occurrences.

5.3.3 Physical model considerations

The physical model should be extended to cover media type which should then be removed from this document.

Rotation properties should also be considered in the context of the physical model and removed from this document. However, application of rotation properties may not be straightforward as rotation is physical behavior and the current physical model focusses (intentionally) on physical inventory. The rotation consideration has similar challenges to temperature and power (both of which overload the physical model).

Similarly, the concept of removable media belongs to the physical model. This could be covered by the equipment in holder structure.

5.3.4 Component-System Pattern

The component-system pattern has been mentioned in this document, but it has not been fully expanded in the model described in this document.

The ComputeConstruct should align more strongly with the component-system pattern and the "..emergent.." associations should be worked further (as there is mapping complexity hidden in these associations). It could be argued that the ViewMappingFunction is necessarily present in the relationship between a system and the apparent emergent component. This should be explored further.

5.3.5 Application of various recursive structure patterns

There are various recursive structures that can be assembled using objects from the models described in TR-512. As shown in this document, the compute model and the software model can be used in a recursive fashion (see section 4 A simple compute example on page 27, which shows a recursion of ComputeConstruct). These structures can give rise to processing constructs, transport functions and control functions which can be assembled to provide network structures that interconnect physical devices that give rise to processing constructs and compute, i.e., can form a larger scale recursion.

These recursive structures can be applied to model real world deployments. Not all recursive structures will appear in real world deployments and some that do appear will not be useful from the perspective of control and management of those deployments.

It will be helpful, in follow-on work, to analyze structures and recursions to identify those that usefully represent real world deployments and to capture these in the form of formally described patterns and in the form of examples.

These patterns and examples can then be used to both inform solutions and to reduce unnecessary variety via emergence of common practice and via standardization which in turn will reduce integration cost/complexity and improve overall efficiency.

5.3.6 Other areas

What units should be defined for memory sizes, for CPU clock speed etc.

Note that:

- Kubernetes works in units of CPU, where "One CPU, in Kubernetes, is equivalent to a *Hyperthread* on a bare-metal Intel processor with Hyperthreading"
- A CPU hardware thread is also called a vCPU (virtual CPU)

Zone size needs to be defined (block, sector, byte…).

The implications of address and data bus limitations needs to be explored.

End of Document