Document Identifier: DSP1071

Date: 2014-11-04

Version: 1.0.0a

Multi-type System Memory Profile

Document Type: Specification

Document Status: Work in Progress

Document Language: en-US

Contents

List of Figures

List of Tables

Foreword

The *Multi-type System Memory Profile* (DSP1071) was prepared by the Server Desktop Mobile Platforms Working Group.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability.

Acknowledgments

The authors wish to acknowledge the following people.

Editor:

- Scott Kirvan – Intel

Contributors:

- John Leung – Intel
- Paul von Behren – Intel
- Barbara Craig -- HP

Introduction

This specification describes a management profile including the CIM model and associated behavior for computer system memory. Specifically, it addresses uni- and multi-processor systems with one or more individually managed memory extents.

The information in this specification should be sufficient for a provider or consumer of this data to unambiguously identify the classes, properties, methods, and values that shall be instantiated to subscribe, advertise, produce, or consume an indication using the DMTF Common Information Model (CIM) Schema.

The target audience for this specification is implementers who are writing CIM-based providers or consumers of management interfaces that represent the components described in this document.

Multi-type System Memory Profile

1 Scope

The Multi-type System Memory Profile extends the management capabilities of referencing profiles by adding the ability to detect and monitor individual memory extents in a computer system. Logical memory extents are modeled in the context of related profiles including those that: 1) model the memory's physical aspects; 2) identify the hosting system; 3) allow for configuration; and 4) define registration information. This profile would generally be used instead of the System Memory Profile (DSP1026) rather than in conjunction with it.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 Approved References

DMTF DSP0004, *CIM Infrastructure Specification 2.7*,
http://www.dmtf.org/standards/published_documents/DSP0004_2.7.pdf

DMTF DSP0215, *Server Management Managed Element Addressing Specification 1.0*,
http://www.dmtf.org/standards/published_documents/DSP0215_1.0.pdf

DMTF DSP0223, *Generic Operations 1.0*,
http://www.dmtf.org/standards/published_documents/DSP0223_1.0.pdf

DMTF DSP0228, *Message Registry XML Schema 1.0*,
http://www.dmtf.org/standards/published_documents/DSP0228_1.0.pdf

DMTF DSP1001, *Management Profile Specification Usage Guide 1.1*,
http://www.dmtf.org/standards/published_documents/DSP1001_1.1.pdf

DMTF DSP1033, *Profile Registration Profile 1.1*,
http://dmtf.org/sites/default/files/standards/documents/DSP1033_1.1.0.pdf

DMTF DSP1011, *Physical Asset Profile*
http://dmtf.org/sites/default/files/standards/documents/DSP1011_1.0.2.pdf

DMTF DSP1022, *CPU Profile*
http://dmtf.org/sites/default/files/standards/documents/DSP1022_1.0.1.pdf

DMTF DSP8016, *WBEM Operations Message Registry 1.0*,
http://schemas.dmtf.org/wbem/messageregistry/1/dsp8016_1.0.xml

DMTF DSP8020, *Message Registry XML Schema Specification 1.0*,
http://www.dmtf.org/standards/published_documents/DSP8020_1.0.xsd

IETF RFC5234, *ABNF: Augmented BNF for Syntax Specifications, January 2008*,
http://tools.ietf.org/html/rfc5234

ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*,
http://isotc.iso.org/livelink/livelink.exe?func=ll&objId=4230456&objAction=browse&sort=subtype

The Open Group, "Regular Expressions" in *The Single UNIX ® Specification, Version 2*, http://www.opengroup.org/onlinepubs/7908799/xbd/re.html

3 Terms and Definitions

3.1

can

used for statements of possibility and capability, whether material, physical, or causal

3.2

cannot

used for statements of possibility and capability, whether material, physical, or causal

3.3

conditional

used to indicate requirements strictly to be followed, in order to conform to the document when the specified conditions are met

3.4

mandatory

used to indicate requirements strictly to be followed, in order to conform to the document and from which no deviation is permitted

3.5

may

used to indicate a course of action permissible within the limits of the document

3.6

memory extent

used generically to indicate a range of memory addresses that can participate in management operations

3.7

memory module

non-technology specific term for a circuit board hosting memory integrated circuits

3.8

need not

used to indicate a course of action permissible within the limits of the document

3.9

optional

used to indicate a course of action permissible within the limits of the document

3.10

persistent memory

byte addressable memory which retains its contents across system power cycles

3.11

referencing profile

indicates a profile that owns the definition of a class used, but not defined, in this document and can be included in the “Referenced Profiles” table

3.12

shall

used to indicate requirements strictly to be followed, in order to conform to the document and from which no deviation is permitted

3.13

shall not

used to indicate requirements strictly to be followed, in order to conform to the document and from which no deviation is permitted

3.14

should

used to indicate that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required

3.15

should not

used to indicate that a certain possibility or course of action is deprecated but not prohibited

3.16

unspecified

indicates that this profile does not define any constraints for the referenced CIM element or operation

4 Symbols and Abbreviated Terms

4.1

NUMA

Non-Uniform Memory Access

4.2

NVM

Non-Volatile Memory

4.3

PM

Persistent Memory

4.4

QoS

Quality of Service

4.5

UMA

Uniform Memory Access

5 Synopsis

Profile Name: *Multi-type System Memory*

Version: 1.0.0a

Organization: DMTF

CIM Schema Version: 2.41

Central Class: CIM_VisibleMemory

Scoping Class: CIM_ComputerSystem

The Multi-type Memory Profile extends the management capabilities of the referencing profiles by adding the capability to represent and manage multiple types of memory within a managed system. The profile supports systems with one or more memory regions where each region can be individually managed.

Table 1 identifies profiles on which this profile has a dependency.

CIM_VisibleMemory shall be the Central Class of this profile.

CIM_ComputerSystem shall be the Scoping Class of this profile. The instance of CIM_ComputerSystem with which the Central Instance is associated through an instance of CIM_SystemDevice shall be the Scoping Instance of this profile.

Table 1 – Related Profiles

Profile Name	Organization	Version	Relationship
Physical Asset	DMTF	1.0.2	Mandatory
Profile Registration	DMTF	1.1.0	Mandatory
CPU	DMTF	1.0.1	Conditional
Memory Configuration Profile	SNIA	1.0.0a	Conditional

6 Description

The Multi-type System Memory Profile describes the elements which allow multiple types of memory to be represented and managed.

This profile can be used to manage the following capabilities of memory regions in a system with multiple types of memory.

- A memory region can have specific quality of service (QoS) characteristics, such a persistence, redundancy, block access.
- A memory region can be configured from a pool of raw memory.
- The characteristics of a memory region can be configured.
- A memory region can be visible to, or have affinity with, specific processors and memory controllers.
- A memory region can be visible to one or more processors (shared).

Figure 1 shows the Multi-type System Memory Profile class hierarchy. For simplicity, the prefix CIM_ has been removed from the names of the classes.

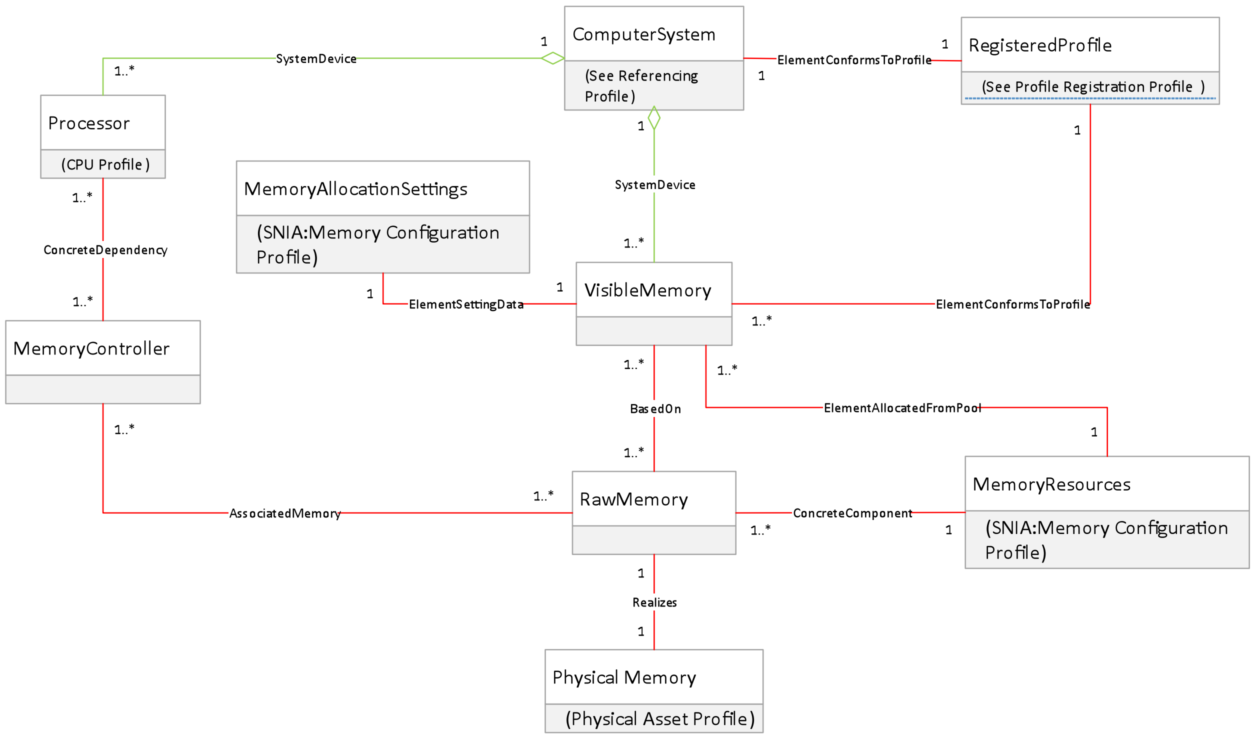

Figure 6-1 – Multi-type System Memory: Class Diagram

Each memory region visible to the computer system is modeled by an instance of CIM_VisibleMemory.

Each physical memory region is associated with its logical counterpart, a raw memory region. Raw memory is not visible to the computer system. Raw memory is modeled by an instance of CIM_RawMemory and its relationship to the visible memory region is modeled by the CIM_BasedOn association.

A memory controller configures raw memory to create the visible memory regions. Memory controllers are represented by instances of CIM_MemoryController and their relationship to the raw memory region is modeled by the CIM_AssociatedMemory association.

In multi-processor systems, memory extents can have an affinity to a specific processor and memory controller. An affinity relationship between memory and a processor/controller can indicate exclusive or preferential access to the memory by that processor. The Multi-type System Memory Profile models a relationship between raw memory extents and their controller and processor such that a management application can determine memory affinity and the physical memory topology.

The SNIA Memory Configuration Profile may be used to model memory regions. That profile includes the CIM_MemoryResources and CIM_MemoryAllocationSetting elements.

The CIM_ElementSettingData and CIM_ElementAllocatedFromPool associations are used to model the relationship between the elements of these two profiles.

7 Implementation

This clause details the requirements related to the arrangement of instances and their most important properties. Class methods are discussed in clause 8; a comprehensive treatment of properties is left to clause 10.

7.1 Representing Raw Memory

An instance of CIM_RawMemory shall represent a memory region which is realized by physical memory, but not visible to the computer system. Instances of CIM_RawMemory shall be associated with an instance of CIM_PhysicalMemory with an instance of CIM_Realizes.

There shall be at least one instance of CIM_RawMemory.

The size given for a CIM_RawMemory instance shall be equal to that given by the SMBIOS Memory Device (type 17) structure for the same memory device.

7.2 Representing Visible Memory

An instance of CIM_VisibleMemory shall represent a memory region which is visible to the computer system. Instances of CIM_VisibleMemory shall be associated with the instance of CIM_ComputerSystem with an instance of CIM_SystemDevice.

There shall be at least one instance of CIM_VisbileMemory. Additional instances of CIM_VisibleMemory may exist when the system contains more than one memory region with distinct memory characteristics. For example, one instance may exist for volatile memory and one for non-volatile memory.

The relationship between the visible memory and the raw memory can be modeled. Each instance of CIM_VisibleMemory shall be associated with one or more instances of CIM_RawMemory, using the CIM_BasedOn association.

7.2.1 CIM_VisibleMemory.HealthState

The CIM_VisibleMemory.HealthState property may have the values 0 (Unknown), 1 (OK) or 2 (Degraded).

7.2.2 CIM_VisibleMemory.EnabledState

The CIM_VisibleMemory.EnabledState property shall have a value of 2 (Enabled) when the visible memory that it represents is visible to the computer system to which it’s scoped.

The CIM_VisibleMemory.EnabledState property shall have a value of 3 (Disabled) when the visible memory, that it represents, is not visible to the computer system to which it’s scoped.

7.2.3 Representing Memory Size

The value of the CIM_VisibleMemory.BlockSize and the CIM_VisibleMemory.NumberOfBlocks properties shall represent the capacity of the memory region visible to the computer system.

The capacity, so represented, shall be the visible (or usable) capacity of the underlying memory extent. For example, memory controllers may support a mirroring feature which has the effect of cutting in half the capacity that is usable by the system. The NumberOfBlocks and BlockSize values shall always take into account (i.e. do not include) space utilized for replication, metadata or the like.

7.2.4 CIM_VisibleMemory.AccessGranularity

The CIM_VisibleMemory.AccessGranularity property shall have a value of 1 (Block Addressable) when the modeled memory region is accessed as a block device. When the memory region is accessed using load and store memory operations the value of CIM_VisibleMemory.AccessGranularity shall be 2 (Byte Addressable). Vendor unique access mechanisms may be represented by values in the vendor reserved range of 32768..65535.

The default value for CIM_VisibleMemory.AccessGranularity shall be 0 (Unknown).

7.2.5 CIM_VisibleMemory.Replication

The CIM_VisibleMemory.Replication property shall indicate whether the contents of the memory region are replicated. The default value for this property shall be 1 (Not Replicated). If the contents are replicated using resources on the local server the value used shall be 2 (Local Replication). If the replicated region exists on a different server (e.g. using RDMA or the like) the value shall be 3 (Remote Replication). Vendor specific replication mechanisms may be represented by values in the vendor reserved range of 32768..65535.

7.3 Representing Topology

Multi-processor systems are common. Often such systems use a Non-Uniform Memory Access (NUMA) configuration in which memory has an “affinity” to a specific processor. In such a system, memory can be accessed optimally by a processor to which it has an affinity; it is more costly (often drastically so) to access from other processors.

In addition to optimal and non-optimal access paths, the topology of memory devices within a system can limit the system’s configuration options. For example a given memory controller may support mirroring between memory address ranges of memory modules under its control. In this case it would be important to understand which memory modules are associated with specific memory controllers. A second example of the importance of topology involves memory interleaving. Memory controllers can enhance overall memory performance by interleaving capacity from multiple memory modules. In a NUMA system it could be advantageous to restrict interleaving to those memory modules with affinity to a specific processor. In this case it would be important to understand the affinity of memory modules for a given processor.

In a uniprocessor system all memory is accessed by a single processor. Conformant implementations include topology information in this degenerate case to minimize special cases for clients attempting to discover memory topology.

7.3.1 CIM_MemoryController

There shall be at least one instance of CIM_MemoryContoller.

An instance of CIM_MemoryController shall be associated to an instance of CIM_RawMemory, which represents raw memory that the memory controller can make available to the computer system, with an instance of CIM_AssociatedMemory.

7.3.2 CIM_Processor

There shall be at least one instance of CIM_Processor, which represents a processor with access to managed memory regions. CIM_Processor instances utilized in this way may be those created by an implementation of the CPU Profile. This is the preferred model. Optionally, CIM_Processor instances may be created specifically for the Multi-type System Memory Profile.

The instance of CIM_Processor shall be associated to the instance of CIM_ComputerSystem, to which the memory is visible, with an instance of CIM_SystemDevice.

7.3.3 Representing Non-Uniform Memory Access Configurations

The instances of CIM_Processor shall be associated to one or more instances of CIM_MemoryController with an instance of CIM_ConcreteDependency.

The instances of CIM_MemoryController shall be associated to one or more instances of CIM_RawMemory with an instance of CIM_AssociatedMemory.

This path from processor to memory controller to raw memory extent describes the NUMA affinity of a given memory extent to a given processor.

Additionally, the CIM_VisibleMemory.ProcessorAffinity property may optionally be used to indicate a preferential relationship between a memory region and a processor. A NUMA relationship is an example of such a preferential relationship. When a NUMA relationship exists between a memory region as modeled by a CIM_VisibleMemory instance and a processor given by CIM_Processor the CIM_VisibleMemory.ProcessorAffinity property is conditionally set to the DeviceID of the processor instance. When no affinity exists or this property is not used it shall be set to an empty string.

When a memory controller has an exclusive or preferential access relationship with a processor this relationship may be represented by setting the CIM_MemoryController.ProcessorAffinity property to the DeviceID of the CIM_Processor instance. When no such relationship exists or the property is not used the CIM_MemoryController.ProcessorAffinity property shall be set to an empty string.

7.4 Representing Memory Configuration

The Multi-type System Memory Profile models the static configuration of memory within a system. For systems that support a configuration process which results in CIM_VisibleMemory instances this profile references the SNIA Memory Configuration Profile, specifically the MemoryAllocationSettings and MemoryResources classes and the associations which link them to the Multi-type System Memory Profile. See Annex B for more information.

8 Methods

This clause details the requirements for supporting intrinsic operations for the CIM elements defined by this profile. No extrinsic methods are defined by this profile.

8.1 CIM_VisibleMemory

Conformant implementations of this profile shall support the operations listed in Table 2 for CIM_VisibleMemory. Each operation shall be supported as defined in DSP0200.

Table 2 – Operations: CIM_VisibleMemory

Operation	Requirement	Messages
GetInstance	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Mandatory	None
ReferenceNames	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.2 CIM_RawMemory

Conformant implementations of this profile shall support the operations listed in Table 3 for the CIM_RawMemory class. Each operation shall be supported as defined in DSP0200.

Table 3 – Operations: CIM_RawMemory

Operation	Requirement	Messages
GetInstance	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Mandatory	None
ReferenceNames	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.3 CIM_MemoryController

Conformant implementations of this profile shall support the operations listed in Table 4 for the CIM_MemoryController class. Each operation shall be supported as defined in DSP0200.

Table 4 – Operations: CIM_MemoryController

Operation	Requirement	Messages
GetInstance	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Mandatory	None
ReferenceNames	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.4 CIM_Processor

Conformant implementations of this profile shall support the operations listed in Table 5 for the CIM_memoryController class. Each operation shall be supported as defined in DSP0200.

Table 5 – Operations: CIM_Processor

Operation	Requirement	Messages
GetInstance	Mandatory	None
Associators	Mandatory	None
AssociatorNames	Mandatory	None
References	Mandatory	None
ReferenceNames	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.5 CIM_ConcreteDependency

Conformant implementations of this profile shall support the operations listed in Table 6 for the CIM_ConcreteDependency class. Each operation shall be supported as defined in DSP0200.

Table 6 – Operations: CIM_ConcreteDependency

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.6 CIM_AssociatedMemory

Conformant implementations of this profile shall support the operations listed in Table 7 for the CIM_AssociatedMemory class. Each operation shall be supported as defined in DSP0200.

Table 7 – Operations: CIM_AssociatedMemory

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

8.7 CIM_BasedOn

Conformant implementations of this profile shall support the operations listed in Table 8 for the CIM_BasedOn class. Each operation shall be supported as defined in DSP0200.

Table 8 – Operations: CIM_BasedOn

Operation	Requirement	Messages
GetInstance	Mandatory	None
EnumerateInstances	Mandatory	None
EnumerateInstanceNames	Mandatory	None

9 Use Cases

This clause contains object diagrams and use cases for the *Multi-type System Memory Profile*.

9.1 Advertising Profile Conformance

Figure 9-1 shows how an instance of CIM_RegisteredProfile is used to indicate the presence of a conforming implementation of the *Multi-type System Memory Profile* and to identify instances of its central class CIM_VisibleMemory.

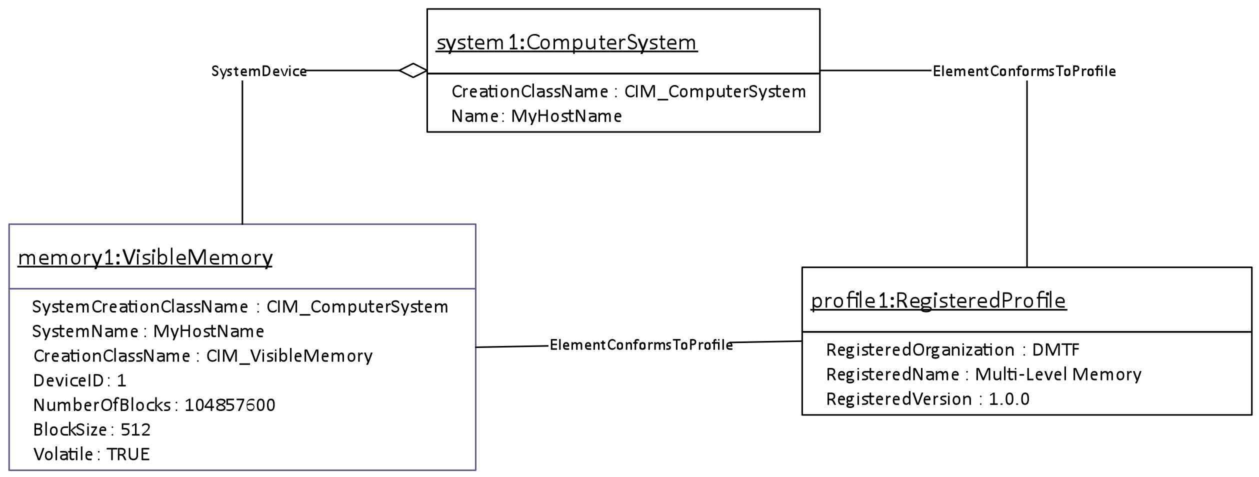
system1:ComputerSystem
CreationClassName : CIM_ComputerSystem
Name: MyHostName
SystemDevice
ElementConformsToProfile
memory1:VisibleMemory
SystemCreationClassName : CIM_ComputerSystem
SystemName : MyHostName
CreationClassName : CIM_VisibleMemory
DeviceID: 1
NumberOfBlocks : 104857600
BlockSize : 512
Volatile : TRUE
ElementConformsToProfile
profile1:RegisteredProfile
RegisteredOrganization : DMTF
RegisteredName : Multi-Level Memory
RegisteredVersion : 1.0.0

Figure 9-1 – Registered Profile

9.2 Single Visible Memory Extent

Figure 9-2 shows the simplest possible configuration with a single memory module (dimm1) contributing its full capacity to a single memory extent (memory1).

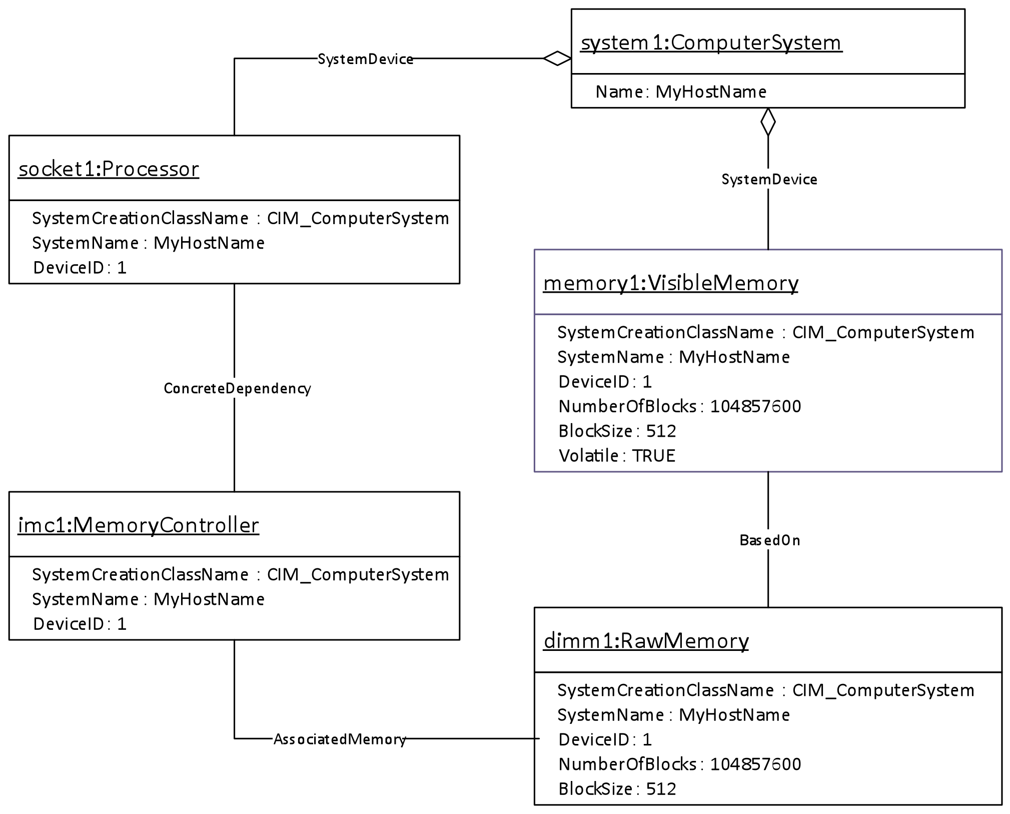

Figure 9-2 Single Visible Memory Extent

9.3 Two Visible Memory Extents

Figure 9-3 models a system configuration in which memory modules and the memory controller support configuring memory address ranges with unique quality of service characteristics. In this example a single memory module has been configured so as to expose two CIM_VisibleMemory extents to the system. The figure shows 1 extent as volatile and the other persistent; the quality of service between the two extents is sufficiently different that one would likely manage and use the extents separately.

Exposing the relationship between CIM_RawMemory and CIM_VisibleMemory extents allows clients to understand reliability and serviceability characteristics of each extent. Clients utilize the CIM_BasedOn association to determine the memory module(s) which host any given CIM_VisibleMemory instance. The position of any given memory module within the system is determined by following the CIM_AssociatedMemory association to the CIM_MemoryController instance.

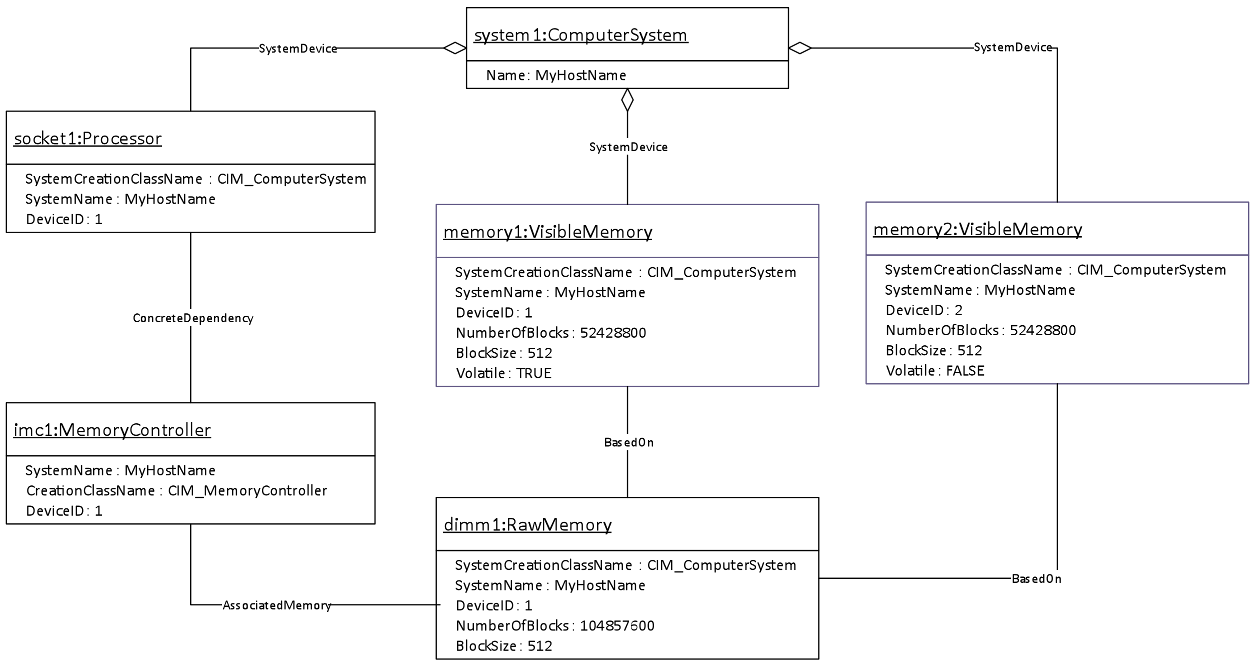

Figure 9-3 – Distinct Visible Memory Extents Object Diagram

9.4 Uniform Memory Access Extents

Figure 9-4 shows a system with a 2 processor UMA architecture. The ProcessorAffinity attribute of the CIM_VisibleMemory instance is set to an empty string indicating no specific affinity. The CIM_RawMemory instance is associated to a CIM_MemoryController which services memory accesses from both CIM_Processor instances. The CIM_MemoryController.ProcessorAffinity attribute is also set to the empty string indicating no affinity to a specific processor.

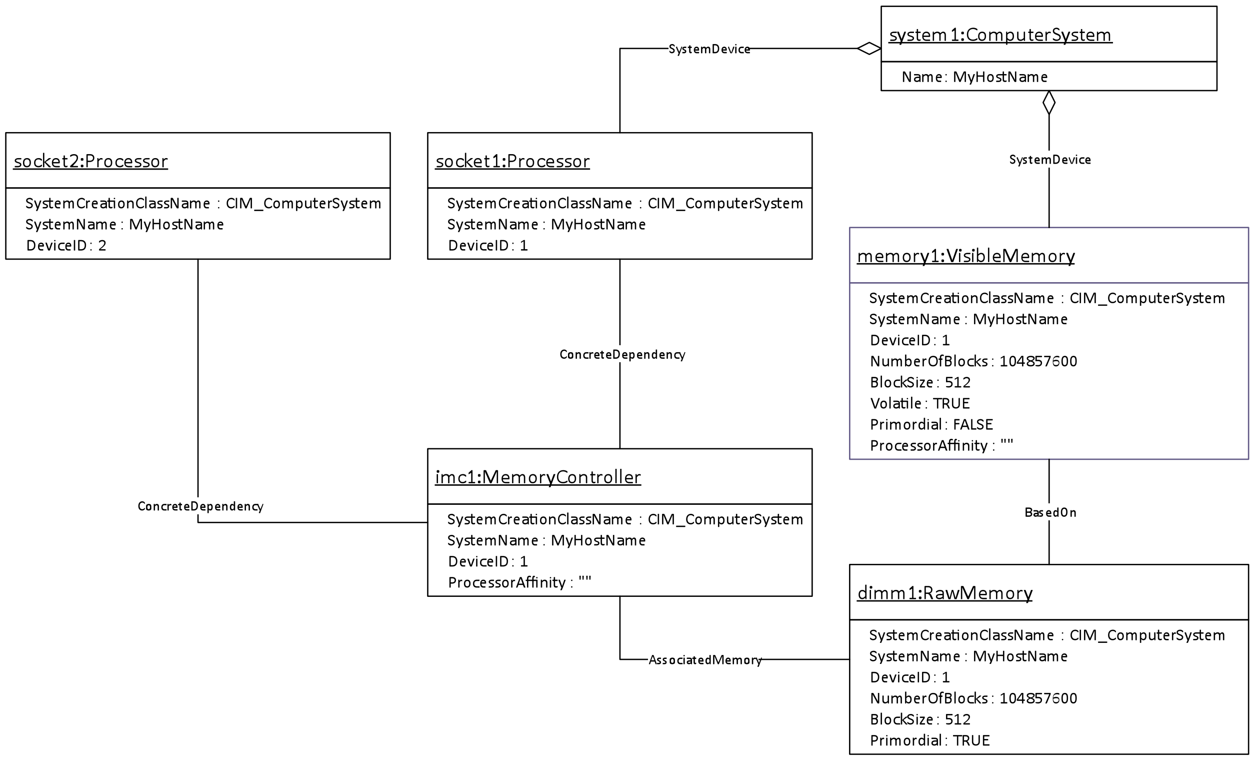

Figure 9-4 --UMA Configuration

9.5 Non-uniform Memory Access (NUMA) Extents

Figure 9-5 shows the model for a multi-processor system with memory extents organized to support NUMA. The CIM_VisibleMemory.ProcessorAffinity property is set to indicate affinity consistent with the results that can be achieved via association traversal (i.e. set to the DeviceID of the affiliated processor). The CIM_MemoryController.ProcessorAffinity is likewise set to the DeviceID of the processor it supports.

In a single processor system (essentially the left or right half of diagram 9-5 in isolation) processor affinity is set to the identity of the only processor.

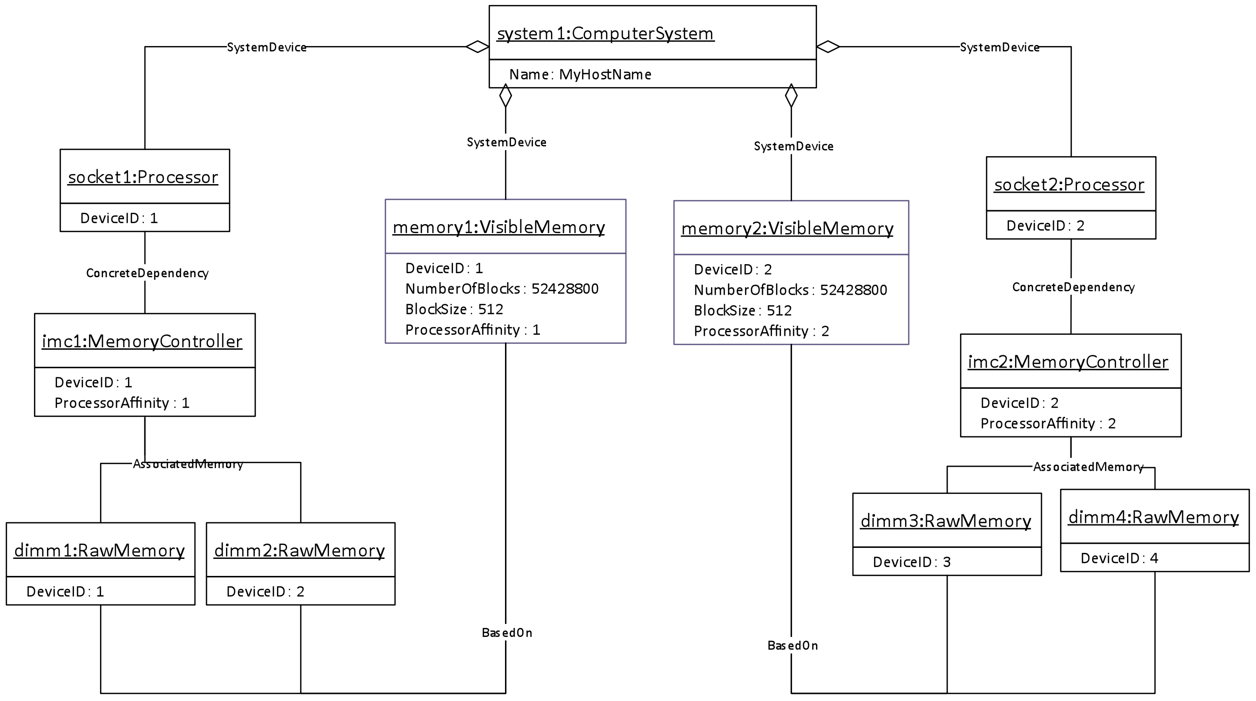

Figure 9-5 – NUMA Configuration Object Diagram

9.6 Determine Persistent Memory Capacity

Determining the capacity of memory with a given QoS is determined by enumerating the CIM_VisibleMemory instances with that QoS and examining the NumberOfBlocks and BlockSize attributes. In figure 9-3 above there are two equally sized instances, one offers volatile memory, the other persistent. Enumerating VisibleMemory instances and summing capacity for those with the Volatile property set to FALSE would give the total memory capacity offering a persistent QoS. Similarly summing the capacity of VisibleMemory instances whose Volatile property is set to TRUE would give the total memory capacity offering a volatile QoS.

9.7 Determine Total Installed Memory Capacity

Total installed memory (in bytes) is calculated by enumerating RawMemory instances and summing the product of NumberOfBlocks and BlockSize.

9.8 Determine Capacity by Processor Affinity

Capacity available to a given processor is determined by following the CIM_ConcreteDependency association to find CIM_MemoryController instances and then following the AssociatedMemory association to CIM_RawMemory instances. Summing the NumberOfBlocks property for the CIM_RawMemory instances, so located, determines the total capacity with an affinity to the selected processor. In figure 9-5, the total capacity with an affinity to the processor in socket 2 is determined by summing the capacity of dimm3 and dimm4.

9.9 Determine Processor Affinity for Visible Memory

Determining whether a given CIM_VisibleMemory instance (assuming the system has a NUMA architecture as given in figure 9-5) has NUMA performance characteristics is determined by following the CIM_BasedOn association to the CIM_RawMemory instances. From there, the CIM_AssociatedMemory association is used to verify that each instance of CIM_RawMemory is controlled by a single processor.

Alternatively, the ProcessorAffinity property maybe sufficient to determine affinity for implementations that utilize it.

10 CIM Elements

Table 9 shows the instances of CIM Elements for this profile. Instances of the following CIM Elements shall be implemented as described in Table 9. Clauses 7 (“Implementation”) and 8 (“Methods”) may impose additional requirements on these elements.

Table 9 CIM Elements – Multi-type System Memory Profile

Element Name	Requirement	Description
CIM_RegisteredProfile	Mandatory	See subclause 10.1
CIM_VisibleMemory	Mandatory	See subclause 10.2, 7.2
CIM_RawMemory	Mandatory	See subclause 10.3, 7.1
CIM_MemoryController	Mandatory	See subclause 10.4, 7.3.1
CIM_Processor	Mandatory	See subclause 10.5, 7.3.2
CIM_ConcreteDependency	Mandatory	See subclause 10.6.
CIM_SystemDevice	Mandatory	See subclause 10.7
CIM_AssociatedMemory	Mandatory	See subclause 10.8
CIM_BasedOn	Mandatory	See subclause 10.9

10.1 CIM_RegisteredProfile

CIM_RegisteredProfile identifies the *Multi-type System Memory Profile* in order for a client to determine whether an instance of CIM_VisibleMemory is conformant with this profile. The CIM_RegisteredProfile class is defined by the *Profile Registration Profile*. With the exception of the mandatory values specified for the properties below, the behavior of the CIM_RegisteredProfile instance is per the *Profile Registration Profile*. Table 10 contains the requirements for elements of this class.

Table 10 – Class: CIM_RegisteredProfile

Elements	Requirement	Notes
RegisteredName	Mandatory	This property shall have a value of "Multi-type System Memory".
RegisteredVersion	Mandatory	This property shall have a value of "1.0.0".
RegisteredOrganization	Mandatory	This property shall have a value of 2 (DMTF).

10.2 CIM_VisibleMemory

The CIM_VisibleMemory class represents memory configured with a given set of QoS attributes. Conformant implementations support attributes as given below.

Table 11 – Class: CIM_VisibleMemory

Elements	Requirement	Notes
CreationClassName	Mandatory	**Key**
DeviceID	Mandatory	**Key**
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
Primordial	Mandatory	**False**
BlockSize	Mandatory	Number of bytes per block. See subclause 7.2.3
NumberOfBlocks	Mandatory	Block count; multiply by BlockSize to get bytes. See subclause 7.2.3.
OperationalStatus	Mandatory	None
HealthState	Mandatory	See subclause 7.2.1
EnabledState	Mandatory	See subclause 7.2.2
Volatile	Optional	None
AccessGranularity	Optional	Access type. See subclause 7.2.4
ProcessorAffinity	Optional	Affiliated processor. See subclause 7.3.3
Replication	Optional	Data replication. See subclause 7.2.5

10.3 CIM_RawMemory

The CIM_RawMemory class represents of the capacity of a given physical memory module. Conformant implementations support attributes as given below.

Table 12 – Class: CIM_RawMemory

Elements	Requirement	Notes
CreationClassName	Mandatory	**Key**
DeviceID	Mandatory	**Key**
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
Primordial	Mandatory	True
BlockSize	Mandatory	Number of bytes per block
NumberOfBlocks	Mandatory	Block count; multiply by BlockSize to get bytes.
OperationalStatus	Mandatory	None
HealthState	Mandatory	None

10.4 CIM_MemoryController

The CIM_MemoryController class represents the controller for one or more raw memory regions. Memory controller modeling is included in this profile to provide an understanding of the system memory topology. Conformant implementations support attributes as given below.

Table 13 – Class: CIM_MemoryController

Elements	Requirement	Notes
CreationClassName	Mandatory	**Key**
DeviceID	Mandatory	**Key**
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
ProtocolSupported	Optional	Identify controller protocol, e.g. DDR3
ProcessorAffinity	Optional	Processor affinity. See subclause 7.3.3

10.5 CIM_Processor

The CIM_Processor class models a processor with access to a visible memory region. This usage of CIM_Processor includes only those properties useful in identifying a processor instance. When implementing both Multi-type System Memory and the CPU Profiles, Multi-type System Memory profile can refer to instances created in accordance with the CPU Profile. When only the Multi-type System Memory profile is implemented the more limited version given below is used. This class is mandatory to remove any ambiguity as to the NUMA/UMA nature of the memory architecture. Conformant implementations support attributes as given below.

Table 14 – Class: CIM_Processor

Elements	Requirement	Notes
CreationClassName	Mandatory	**Key**
DeviceID	Mandatory	**Key**
SystemCreationClassName	Mandatory	**Key**
SystemName	Mandatory	**Key**
Family	Optional	This property supported if it can be used to determine processor support for specific memory management features.
OtherFamilyDescription	Conditional	Used if Family value is “1”.
Stepping	Optional	This property supported if it can be used to determine processor support for specific memory management features.
OtherIdentifyingInfo	Optional	This property supported if it can be used to determine processor support for specific memory management features. Recommended values: Processor Type, Processor Model, and Processor Manufacturer.
IdentifyingDescriptions	Conditional	If OtherIdentifyingInfo is used.

10.6 CIM_ConcreteDependency

The CIM_ConcreteDependency association is used to relate an instance of CIM_MemoryController to a CIM_Processor instance. Table 15 contains the requirements for elements of this class.

Table 15 – Class: CIM_ConcreteDependency

Elements	Requirement	Notes
Antecedent	Mandatory	This property shall be a reference to an instance of the CIM_Processor class. Cardinality is "1..*".
Dependency	Mandatory	This property shall be a reference to an instance of a concrete subclass of the CIM_MemoryController class. Cardinality is "1..*".

10.7 CIM_SystemDevice

10.7.1 Relating CIM_Processor to CIM_ComputerSystem

CIM_SystemDevice association is used to relate an instance of CIM_Processor with an instance of CIM_ComputerSystem. Table 16 contains the requirements for elements of this class.

Table 16 – Class: CIM_SystemDevice –use 1

Elements	Requirement	Notes
GroupComponent	Mandatory	This property shall be a reference to an instance of CIM_ComputerSystem. Cardinality is "1".
PartComponent	Mandatory	This property shall be a reference to an instance of CIM_Processor. Cardinality is "1..*".

10.7.2 Relating CIM_VisibleMemory to CIM_ComputerSystem

CIM_SystemDevice association is used to relate an instance of CIM_VisibleMemory with an instance of CIM_ComputerSystem. Table 16 contains the requirements for elements of this class.

Table 17 – Class: CIM_SystemDevice –use 2

Elements	Requirement	Notes
GroupComponent	Mandatory	This property shall be a reference to an instance of CIM_ComputerSystem. Cardinality is "1".
PartComponent	Mandatory	This property shall be a reference to an instance of CIM_VisibleMemory. Cardinality is "1..*".

10.8 CIM_AssociatedMemory

The CIM_AssociatedMemory association is used to relate the CIM_MemoryController instance to the CIM_RawMemory instance to which it applies. Table 18 contains the requirements for elements of this class.

Table 18 – Class: CIM_AssociatedMemory

Elements	Requirement	Notes
Antecedent	Mandatory	This property shall be a reference to an instance of the CIM_RawMemory class. Cardinality is "1..*".
Dependent	Mandatory	This property shall be a reference to an instance of the CIM_MemoryController class. Cardinality is "1..*".

10.9 CIM_BasedOn

The CIM_BasedOn association is used to relate the CIM_VisibleMemory to the CIM_RawMemory on which it is hosted. Table 19 contains the requirements for elements of this class.

Table 19 – Class: CIM_BasedOn

Elements	Requirement	Notes
Antecedent	Mandatory	This property shall be a reference to an instance of the CIM_RawMemory class. Cardinality is "1".
Dependent	Mandatory	This property shall be a reference to an instance of the CIM_VisibleMemory. Cardinality is "1".

ANNEX A
(informative)

Change Log

Version	Date	Description
1.0.0a	9/29/2014	Draft Standard

ANNEX B

SNIA Memory Configuration Profile

This profile, the Multi-type System Memory Profile is being pursued with the DMTF while a closely related profile tentatively named the *Memory Configuration Profile* is being pursued with SNIA. Since memory management has been the purview of the DMTF it was felt that the static view defined by the Multi-type System Memory Profile was best pursued with the DMTF as a follow-on to the existing System Memory Profile. The management of memory configuration is being pursued with SNIA for similar reasons, its similarity to existing SNIA profiles and the blurring of the typical roles played by memory and storage. Indeed, the primary motivation for updating memory management profiles at this time is the recent introduction of non-volatile memory technologies that use typical memory form factors (e.g. DIMM) and typical memory interconnects (e.g. DDR3) but have features/characteristics usually associated with storage.

The SNIA Memory Configuration Profile is conceived as building upon the Multi-type System Memory Profile. As such its detailed definition is trailing the definition provided in this document. That said, some high-level definition has occurred and may be useful in putting the Multi-type System Memory Profile in context. Figure B-1 below identifies key classes in the Memory Configuration Profile focusing on those that associate with Multi-type System Memory Profile classes.

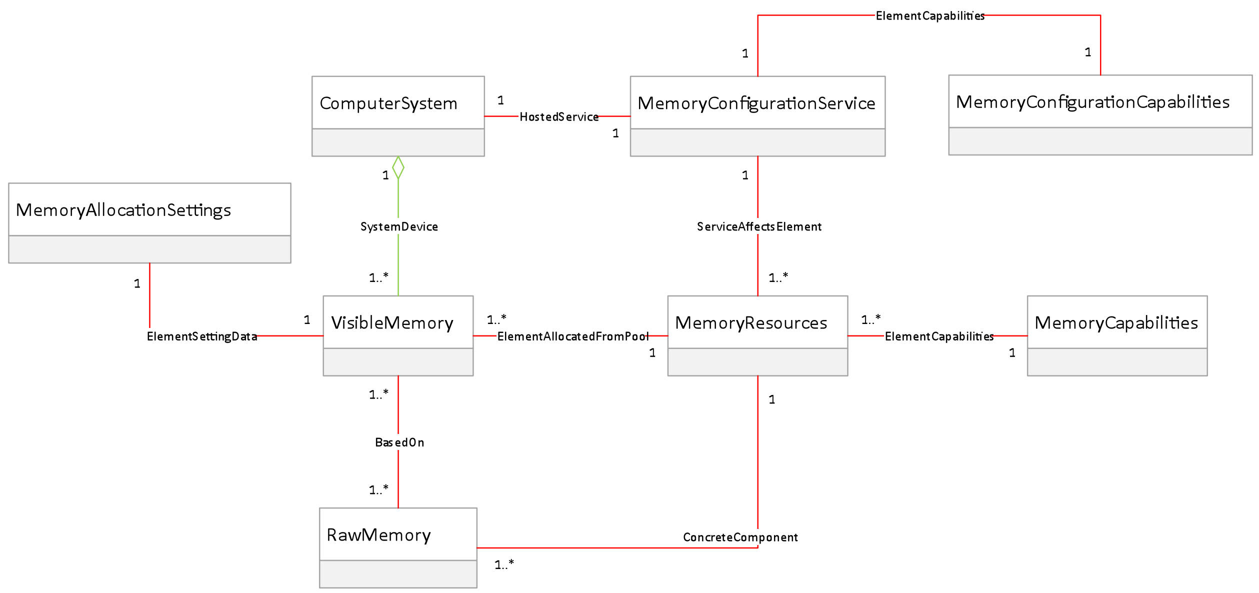

Figure B-1 Memory Configuration Profile

- **ComputerSystem** –from the referencing profile
- **VisibleMemory** –the central class of the Multi-type System Memory Profile. A system visible memory resource.
- **RawMemory** –referenced from the Multi-type System Memory Profile, a primordial memory extent associated with a specific memory module.
- **MemoryAllocationSettings** –the settings provided during the provisioning process that resulted in a given VisibleMemory instance. Also used as input to the provisioning extrinsic method.
- **MemoryAllocationService** –provides extrinsic methods for memory configuration. These methods result in the allocation or return of resources to the MemoryResources pool and the creation or destruction of VisibleMemory instances.

- **MemoryConfigurationCapabilities** –describes the supported extrinsic method support available from the MemoryAllocationService.
- **MemoryCapabilities** –describes the configurable features of the resources aggregated under the MemoryResources pool.