OCP GPU & Accelerator Management Interfaces

Version 0.5

Authors: (in alphabetical order)

David Blocker, NVIDIA

James Bodner, NVIDIA

Deepak Kodihalli, NVIDIA

Choudary Maddukuri, Microsoft

Linda Wu, NVIDIA

Justin York, Google

Executive Summary

Management of GPUs is not standardized, resulting in significant effort and time to onboard each new HW design. The lack of standardization is also a burden on suppliers who must accommodate varying requirements from their customers. This document describes industry standard formats and protocols that make it easier for

CSPs (Cloud Service Providers) to onboard new GPU and accelerator designs with less toil and faster time to market while reducing manageability permutations for suppliers.

Table of Contents

Table of Contents

Executive Summary **1**

Table of Contents **2**

1 Introduction **4**

2 Overview **4**

2.1 Discrete Accelerator Devices Manageability *4*

2.1.1 Static discovery 5

2.1.2 Transport Protocol 5

2.1.3 Attestation 5

2.1.4 Management capability discovery 5

2.1.5 Base monitoring and control 5

2.1.6 Firmware management 6

2.2 UBB Accelerator Devices Manageability *6*

2.2.1 Static discovery 6

2.2.2 Attestation 6

2.2.3 Base monitoring and control 6

2.2.4 Firmware management 6

3 Discrete Accelerator Device Interfaces **7**

3.1 Access Frequency *7*

3.2 FRU *7*

3.3 MCTP *8*

3.3.1 Bus Owner and EIDs 8

3.3.2 MCTP Discovery 8

3.3.3 Bridge Routing Table Initialization: 11
3.3.4 Hot Join flow: 12
3.3.5 MCTP command details 13

3.4 SPDM 14
3.4.1 Required SPDM Commands 14
3.4.2 Required Capabilities for SPDM 15
3.4.3 Required Algorithms for SPDM 15

3.5 PLDM 15
3.5.1 Discrete GPU/Accelerator Modules 16
3.5.2 PLDM Sensors & Effectors 16
3.5.3 Entity Association 17
3.5.4 Sensor & Effector Entity Mapping 20
3.5.5 PLDM Type IDs 24

3.6 MCTP OEM VDM 25
3.6.1 Management Protocol Structures 25
3.6.2 Completion Codes 28
3.6.3 OEM Events 29

4 UBB Accelerator Device Interfaces 29

4.1 Redfish 29
4.1.1 Redfish Tree 29
4.1.2 Location Objects 30
4.1.3 RAS Error Injection 31

4.2 Out-of-Band Interfaces (MCTP, PLDM, and SPDM) 36

5 Conclusion 36

6 Glossary 36

7 References 36

8 License - Open Web Foundation (OWF) CLA 37

9 OCP Tenets 38

10 About Open Compute Foundation 39

1 Introduction

Advances in accelerator-based workloads are driving the need for ever-faster bring-up and deployment of new accelerator-based system designs. Different accelerator components from different vendors implement a wide variety of management interfaces with little commonality. This lack of commonality causes extra toil and increases time-to-market for deploying new, innovative designs.

This document describes a common and generic framework for managing GPU and accelerators by way of management interfaces designed using standards-based management protocols.

This document covers both discrete accelerator devices (e.g., PCI-e CEM cards) and Universal Baseboard (UBB) designs (e.g., OCP OAI HGX).

2 Overview

The section specifies the interfaces for managing both Discrete and UUB design GPU/Accelerator's.

2.1 Discrete Accelerator Devices Manageability

Discrete accelerator devices are GPUs packaged as standard CEM form-factor PCI-e adapter. Discrete accelerator device management can be broken down into several categories, each of which tie-in to various industry standards.

Table 1 – DMTF Standard Protocols

Manageability Objective	Technology	Standard
Static discovery	IPMI FRU	Platform Management FRU Information Storage Definition (rev. 1.3)
Transport protocol	MCTP	DMTF DSP0236 revision 1.3.1 or later
Attestation	SPDM	DMTF DSP0274 revision 1.2.1 or later
Management capability discovery	PLDM Type 0	DMTF DSP0240 revision 1.1.0 or later
Base monitoring and control	PLDM Type 2	DMTF DSP0248 revision 1.2.2 or later
Accelerator monitoring and control	PLDM Type 2	*WIP* DMTF DSP2061
Firmware management	PLDM Type 5	DMTF DSP0267 revision 1.2.0 or later
File I/O	PLDM Type (Future/TBD)	*WIP* DMTF DSP0242

2.1.1 Static discovery

The initial, static discovery of devices seeks to obtain primarily immutable information about the discrete accelerator device. This is done by reading the contents of a dedicated FRU EEPROM contained within the device.

2.1.2 Transport Protocol

MCTP is the media independent protocol for communication among management controllers within system. This section below describes the Bus owner relationship, the endpoint discovery, EID assignment, the bridges and the EID pool assignments and corresponding flows.

2.1.3 Attestation

Once static and MCTP endpoint discovery is complete, an important step is attestation of the device. SPDM (DSP0274) operates over MCTP as a standardized mechanism to authenticate hardware identity and measure firmware identity. Using SPDM, the management controller serves as an SPDM requestor and discovers and negotiates the security capabilities to be used by the responder (discrete accelerator device). The management controller then authenticates the responder. Lastly, the management controller requests firmware measurements from the responder.

Note: SPDM attestation may happen repeatedly at the discretion of the management controller and is not limited to when the discrete accelerator device is initially discovered.

2.1.4 Management capability discovery

Once the hardware discrete accelerator device has been authenticated and measured, continuous management begins by discovering the management capabilities of the device. For standards-based management, subsequent discovery of the device capabilities happens by way of PLDM Type 0 (base) discovery.

2.1.5 Base monitoring and control

Once basic manageability capabilities are discovered dynamically from the device, continuous monitoring and control is performed by way of PLDM Type 2 (DSP0248).

2.1.6 Firmware management

Standards-based device firmware management requires support for PLDM Type 5 (DSP0267). DSP0267 provides a standardized way to query the version of the device's firmware (inventory) and to push firmware to the device (update).

2.2 UBB Accelerator Devices Manageability

Universal baseboard (UBB) designs are a multi-GPU/accelerator devices with high-speed intra-GPU interconnects, PCI-e switches and retimers, and complex topologies. In those HW devices the expectation is the UBB must have its own management controller (subsequently referred to as the UBBMC) with a standard Redfish interface exposed via a network connection and also DMTF MCTP & PLDM for high-frequency telemetry (e.g. thermal loop data) via sideband connection (I2C/I3C).

Table 2 – DMTF Standard Protocols

Manageability Objective	Technology	Description
Static discovery	IPMI FRU	Platform Management FRU Information Storage Definition (rev. 1.3)
Transport protocol	Redfish/MCTP	Redfish via NW and MCTP via I2C/I3C
Attestation	SPDM	Redfish and I2C/MCTP
Accelerator monitoring and control	Redfish/PLDM Type 2	Comprehensive monitoring via Redfish and High frequency telemetry via I2C/I3C
Firmware management	Redfish	Via Redfish

2.2.1 Static discovery

UBB accelerator designs are expected to present a FRU EEPROM (identical to what is required for discrete accelerator devices) and a UBBMC with a Redfish and limited MCTP and PLDM interface.

2.2.2 Attestation

SPDM attestation is managed by the BMC but control is effected through a UBBMC Redfish interface.

2.2.3 Base monitoring and control

UBB accelerator designs are managed predominantly through Redfish implemented by the UBB BMC. High frequency and critical telemetry like power and thermal are managed via MCTP/PLDM through a sideband connection like I2C/I3C.

2.2.4 Firmware management

UBB accelerator designs shall support firmware update through Redfish implemented by the UBBMC.

3 Discrete Accelerator Device Interfaces

This section lists the minimum set of protocols and OEM extension/format that any GPU/accelerator vendor needs to support on their discrete accelerator devices for hyperscalers to seamlessly manage these devices without any additional vendor/device specific work.

3.1 Access Frequency

The following table provides a high-level mapping of Data/Telemetry categories to protocols including the required frequency of access that must be reliably supported.

Table 3 – Discrete Accelerator Management Protocols

Telemetry Data/Attribute (unordered)	Protocol	Use Case	Frequency
Monitoring (Sensors)	PLDM Type 2	e.g. Thermal, power…	1s
Inventory	MCTP/PLDM		On demand
Discovery	MCTP		On demand
Support Dump	Future direction: PLDM Type (DSP0242 - WIP)	Comprehensive triage	On demand
Debug Logs	Future direction: PLDM Type (DSP0242 - WIP)		On demand
Updates	PLDM Type 5		
Counters	Future direction: PLDM and MCTP OCP OEM	Tuning	"seconds"
Configuration/Settings	PLDM Type 2 (Effectors)		On demand

3.2 FRU

Static discovery of the discrete accelerator device is accomplished by accessing a FRU EEPROM.

The discrete accelerator device shall support an i2c/i3c-based EEPROM that carries and IPMI FRU formatted data supporting the following requirements:

- Min 512bytes, ideally 4k (512byte min forces 2byte offsets)
- Internal Use Area: Optional
- Chassis Info Area: Optional
- Board Info Area: Mandatory (see below)
- Product info area: Mandatory (80 Bytes Minimum
- Multi Record Area: Mandatory for any OEM extensions and the last area so that it extends.

It is mandatory that the FRU EEPROM device be a dedicated secondary device on the bus and not be subject to multiple primary controllers (e.g., one primary on the CEM card and the other primary on the host board.) Multi-primary designs are difficult to reliably marshal access to, reducing system reliability.

The following table lists FRU areas that must be populated.

Table 4 – IPMI FRU Fields

FRU Area	Fields	Required w/ Valid Value
Common Header	Common Header	N
Chassis Info Area	Chassis Type	N
	Chassis Part Number	N
	Chassis Serial Number	N
Board Info Area	Mfg Date/Time	N
	Board Manufacturer	Y
	Board Product Name	Y
	Board Serial Number	Y
	Board Part Number	Y
Product Info Area	Manufacturer Name	Y
	Product Name	Y
	Product Part/Model Number	Y
	Product Version	N
	Product Serial Number	Y
Multi-Record Area	Mandatory for any OEM extensions and the last area so that it extends.	Y, if applicable

3.3 MCTP

This section describes the MCTP discovery and inventory flows and lists minimum required commands. This section also proposes and defines a high-level generic VDM command format for data that needs OEM extensions.

3.3.1 Bus Owner and EIDs

The hyperscaler's BMC is the topmost bus owner and all other devices are endpoint devices. The endpoint devices can themselves be a bus owner when they are connected to 2 or more MCTP buses (also referred to as a "bridged device". The "topmost bus owner" is responsible for allocating ALL EIDs. It assigns a pool of EIDs to the bridge devices so that they allocate those EIDs to the device behind them.

3.3.2 MCTP Discovery

MCTP discovery/re-discovery happens on the following conditions:

- AC power ON
- DC power Cycle
- BMC reset
- Discovery Notify command

Following an AC/DC power ON, the discovery can occur any time during the system boot and typically complete before the host operating system has booted. The end device should be ready in less than 60sec from power ON and the PCI-e enumeration must not impact the I2C/I3C communication.

The flow charts below detail the MCTP discovery and EID assignment flows.

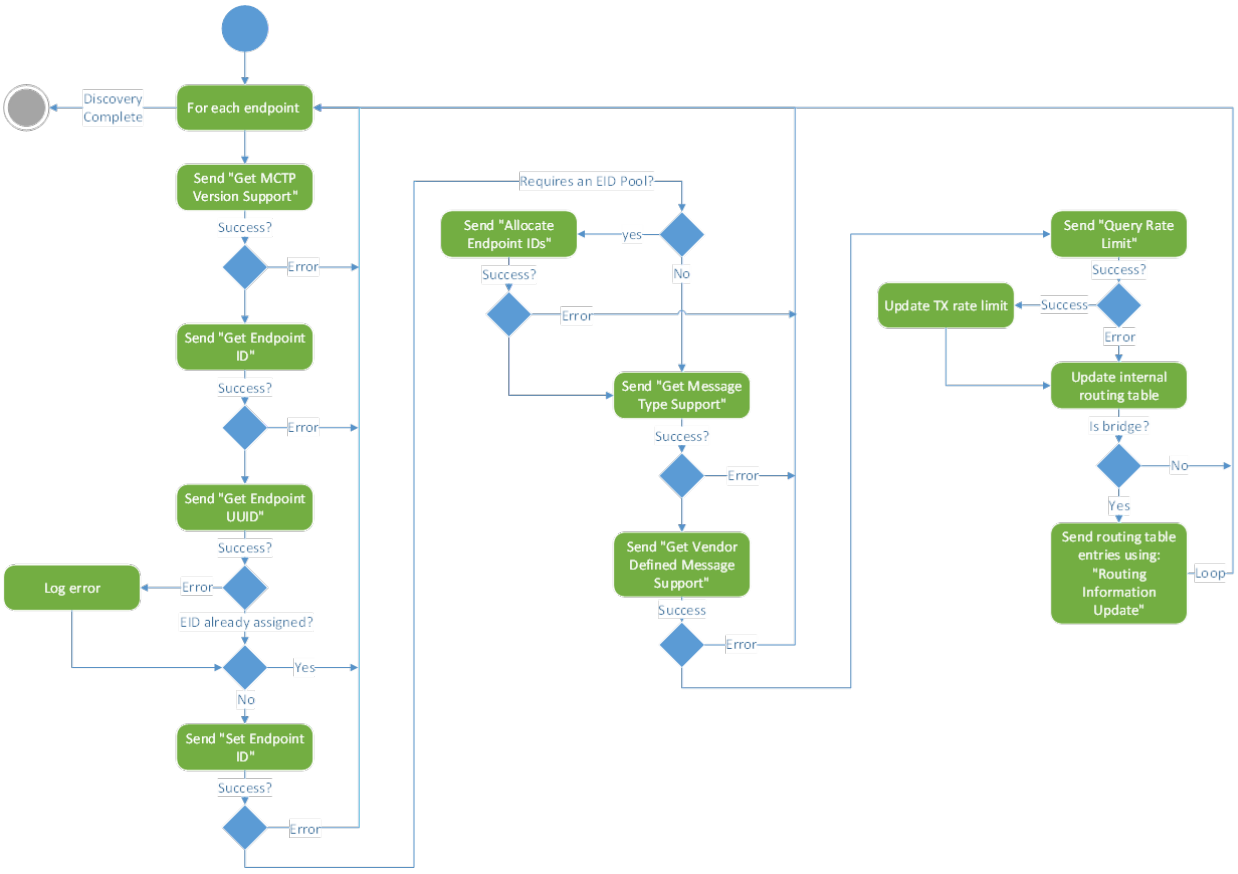

Figure 1 – MCTP Discovery Flow

3.3.3 Bridge Routing Table Initialization:

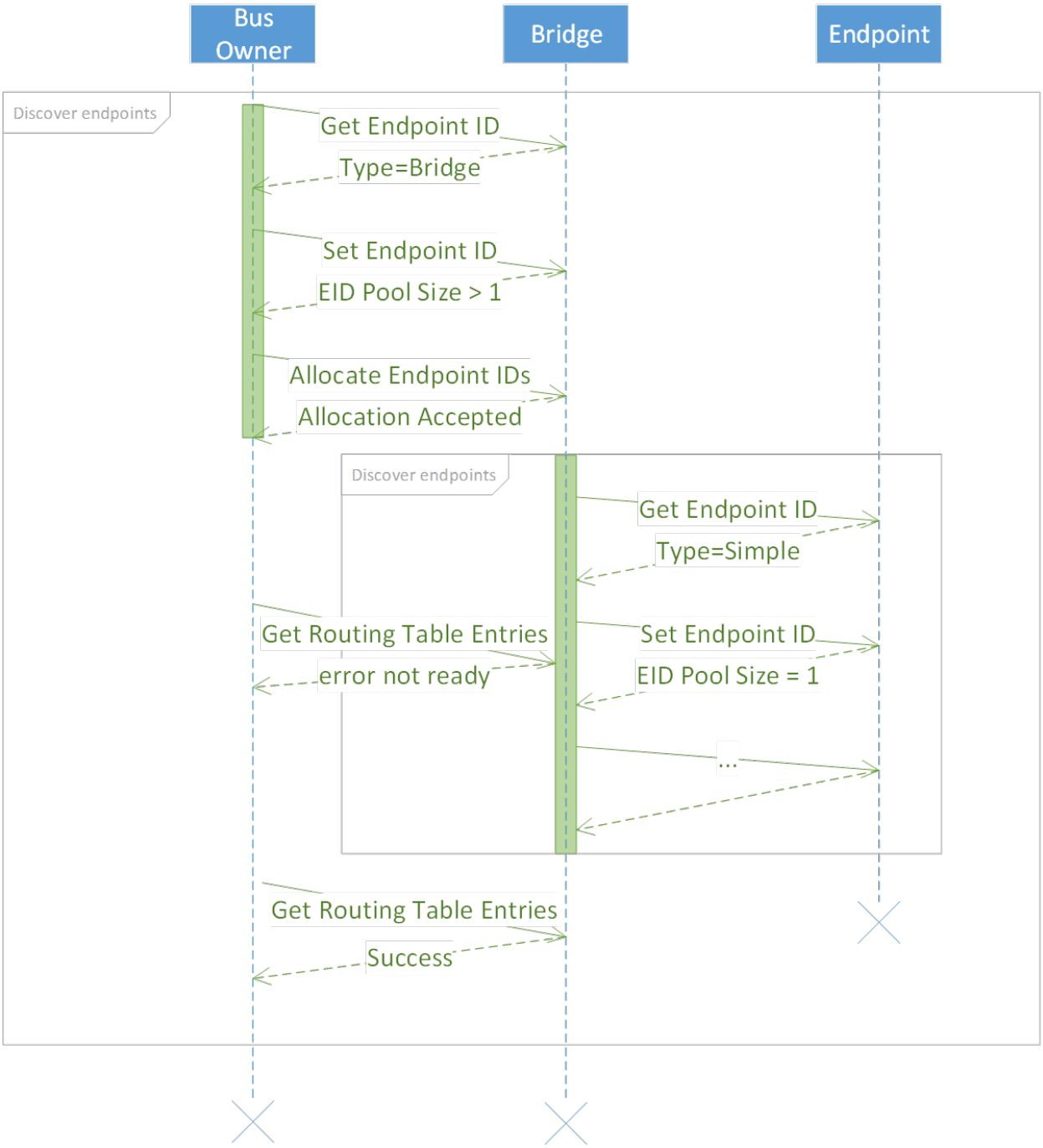

Figure 2 – MCTP Bridge Routing Flow

3.3.4 Hot Join flow:

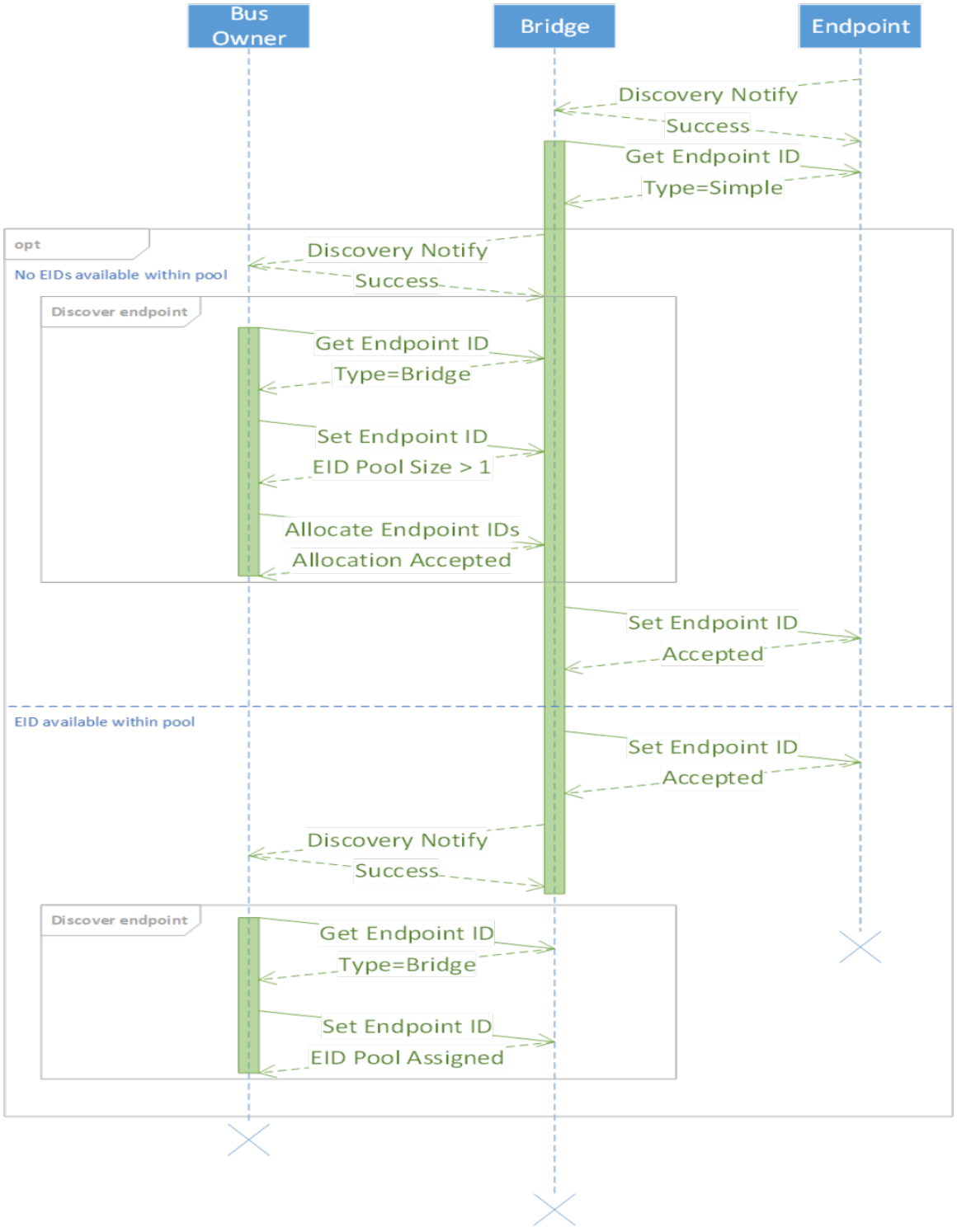

Figure 3 – MCTP Hot Join Flow

3.3.5 MCTP command details

Table 4 lists the MCTP control commands and if they are required for use. This table is very similar to Table 12 within the DSP0236 specification with deviations highlighted in yellow.

Table 5 – MCTP Commands

Cmd Code	Command Name	Endpoint		Bridge		Notes
0x01	Set Endpoint ID	**Ma**	*Ng*	**Ma**	**Mg**	
0x02	Get Endpoint ID	**Ma**	Og	**Ma**	**Mg**	
0x03	Get Endpoint UUID	**Ca**[1]	Og	**Ca**[1]	Og	1. Mandatory if device connects to multiple buses.
0x04	Get MCTP Version Support	**Ma**	Og	**Ma**	**Mg**[1]	1. Bridges must use this command to verify compatible MCTP control protocol versions.
0x05	Get Message Type Support	**Ma**	Og	**Ma**	Og	
0x06	Get Vendor Defined Message Support	**Ca**[1]	Og	**Ca**[1]	Og	1. Mandatory to accept if device used vendor defined messaging.
0x07	Resolve Endpoint ID	*Na*	Og	**Ma**	Og	
0x08	Allocate Endpoint IDs	*Na*	Ng	**Ma**	*Ng*	
0x09	Routing Information Update	Oa	Og	**Ma**	*Ng*	
0x0A	Get Routing Table Entries	*Na*	Og	**Ma**	Og	
0x0B	Prepare for Endpoint Discovery	**Ca**[1]	*Ng*	**Ca**[1]	Ng	1. Mandatory on a per bus basis to support endpoint discovery.
0x0C	Endpoint Discovery	**Ca**[1]	*Ng*	**Ca**[1]	Ng	1. Mandatory on a per bus basis to support endpoint discovery.
0x0D	Discovery Notify	*Na*	**Cg**[1]	**Ca**[1]	Cg[1]	1. Mandatory if physical binding supports hot joins (e.g. I3C and PCI VDM).
0x0E	Get Network ID	Oa	Og	Oa	Og	
0x0F	Query Hop	*Na*	Og	**Ma**	Og	
0x10	Resolve UUID	*Na*	Og	**Ca**[1]	Og	1. Mandatory if device and/or any connected devices connects to multiple buses.
0x11	Query rate limit	Ca[1]	Og	Ca[1]	Og	1. Mandatory if device can support more than 1 request at a time.

0x12	Request TX rate limit	Oa	Og	**Ma**	Og	1. Bridges must allow for endpoints to change their rate limits
0x13	Update rate limit	Oa	Og	Oa	Og	
0x14	Query Supported Interfaces	Oa	Og	Oa	Og	

Key for Endpoint/Bridge columns:
Ma = mandatory to accept
Mg = mandatory to generate
Oa = optional to accept
Og = optional to generate
Ca = conditional to accept (see notes)
Cg = conditional to generate (see notes)
Na = not applicable to accept
Ng = not applicable to generate

3.4 SPDM

The OCP profile for accelerator attestation describes the set of SPDM commands, capabilities, ciphers, and hash algorithms required to deliver a GPU/accelerator that complies with OCP security requirements.

This profile will ultimately exist in a separate publication from the OCP Security Workgroup (Attestation of System Components). That document is in a pre-publication state so the content is included here for review purposes.

- Accelerator attestor devices MUST support the SPDM standard and requirements below to be compliant with the "OCP Attestation SPDM Profile for Accelerators"
- Attestor devices MUST conform to the set of capabilities as defined in the table "Required Capabilities for SPDM".
- Attestor devices should support the current SPDM version and MUST support version 1.1 or higher

3.4.1 Required SPDM Commands

Table 6 – SPDM Minimum Required Commands

SPDM Version	Command	Required
1.0/1.1	GET_VERSION	Y
	GET_CAPABILITIES	Y
	NEGOTIATE_ALGORITHMS	Y
	GET_DIGESTS	Y
	GET_CERTIFICATE	Y
	CHALLENGE	Y
	GET_MEASUREMENTS	Y
	GET_CSR	Y

1.2 (Recommended)	SET_CERTIFICATE	Y
	CHUNK_SEND	Y
	CHUNK_GET	Y

3.4.2 Required Capabilities for SPDM

The following table lists the SPDM capabilities as defined in the CAPABILITIES response that are required for attestor devices that to be compliant with the "OCP Attestation SPDM Profile for Accelerators"

Table 7 – SPDM Required Capabilities

Required Capability	Notes
CERT_CAP	GET_DIGESTS and GET_CERTIFICATE
CHAL_CAP	CHALLENGE
MEAS_CAP	10b (Can measure and generate signatures)
MEAS_FRESH_CAP	1 (Always return fresh measurements)

3.4.3 Required Algorithms for SPDM

Attestor devices are allowed a large number of algorithm combinations under the SPDM specification. To simplify capability matching, the attestor devices must follow the guidelines in the following tables.

Table 8 – SPDM Minimum Required Algorithms

Algorithm Type	Recommended Capability
Asymmetric	TPM_ALG_ECDSA_ECC_NIST_P521
Measurement Hash	TPM_ALG_SHA_512
	TPM_ALG_SHA3_384
	TPM_ALG_SHA3_512

3.5 PLDM

This section describes how to model a GPU/accelerator PCI-e adapter using PLDM Monitoring and Control protocol. It provides general guidelines on how to model the adapters and their components so that the CSPs can implement a common and generic framework for monitoring, control, and component firmware updates across devices and vendors.

The guidelines are based on the PLDM standards as define by the DSP0248_1.1.0.

3.5.1 Discrete GPU/Accelerator Modules

The block diagram below is a generic representation of an accelerator adapter and can vary based on the vendor. Here, the discrete accelerator device represents the overall adapter and enclosed components as follows:

- SMC: This is the management controller (Satellite Management Controller) which implements PLDM and communicates to the Host BMC.
- GPUx: One or more GPU/accelerator controllers
- InterLink: one or more controllers that enable a high-speed connection between GPUs
- PCI-e switch/controller

The block diagram also lists a subset of sensors/effectors that are supported by various components. The following sections describe how to model various sensors and controls via PLDM sensors, effectors and corresponding PDR's.

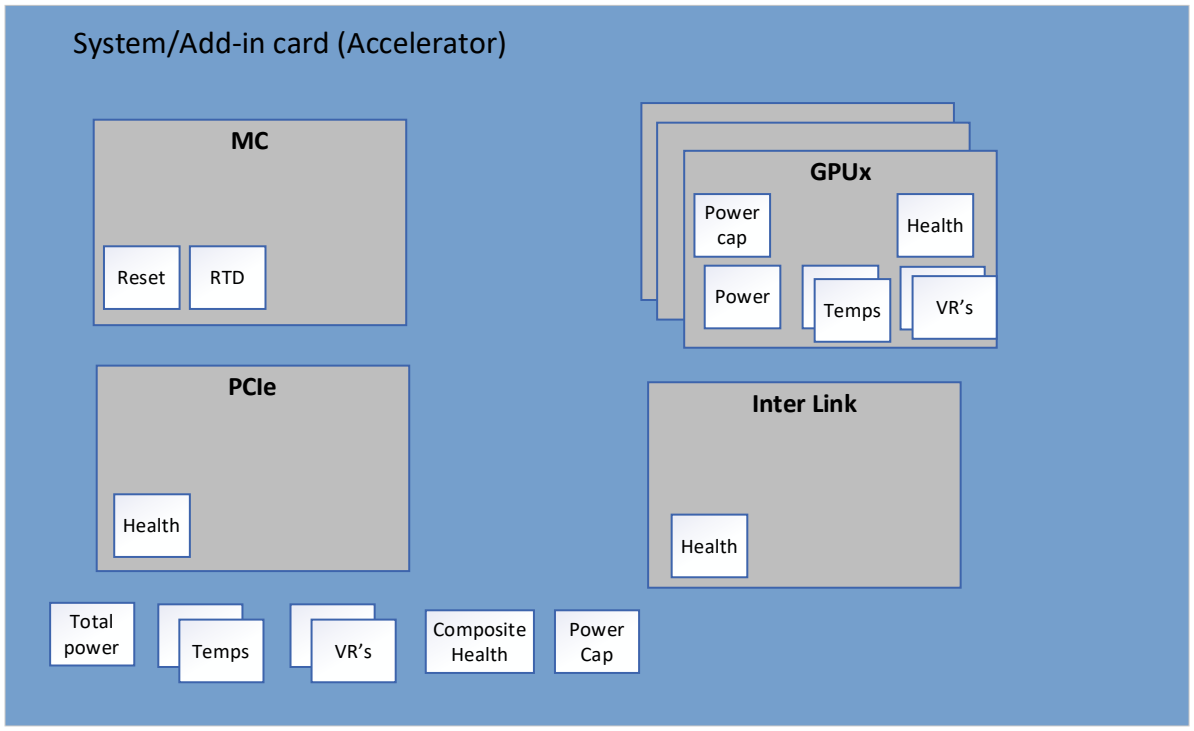

Figure 4 – PCIe CEM card

3.5.2 PLDM Sensors & Effectors

The section below provides high-level details of sensors and effectors and presents a minimum set of requirements.

3.5.2.1 Sensors

The implementation must support both numeric and state sensors. At a minimum the sensors listed below are required if present in the hardware design.

The temperature and power sensors are standard numeric sensors supported by the modules whereas the composite health sensor represents the overall health of the adapter.

Table 9 – PLDM Sensors and Types

Sensors	Sensor Type	Add-in Card	MC	GPU	Link	PCIe
Temp	Numeric	•		•	•	
Power	Numeric	•		•		
Total power	Numeric	•				
VR's	Numeric	•		•		
Composite Health	State	•				
Health	State	•	•	•	•	•

3.5.2.2 Effectors

The implementation must support both numeric and state effectors. At a minimum, the effectors listed below are required if supported by the hardware design.

The power cap effector must be supported at the discrete accelerator device (adapter) level and is optional at the GPU/accelerator module. The reset and reset to defaults (RTD) effectors must be supported by the management controller.

Table 10– PLDM Effectors

Effectors	Add-in Card	MC	GPU	Link	PCIe
Power Cap	•		•		
RTD		•			
Reset		•			

3.5.3 Entity Association

The following section describes how to model the entity association PDRs. These PDRs represent modules in a hierarchical model. The sections describe both physical and logical associations.

3.5.3.1 Physical Entity Association

These physical association PDRs are used to describe physical components and their association. The entity association PDR uses the following references to describe the modules and their association as defined in DSP0248:

- Container ID (an opaque number)
- Contained ID (an opaque number)
- Entity type: The entity type ID are as defined in DSP0248
- Entity instance: Module instance such as “GPU 0” or “GPU 1”.

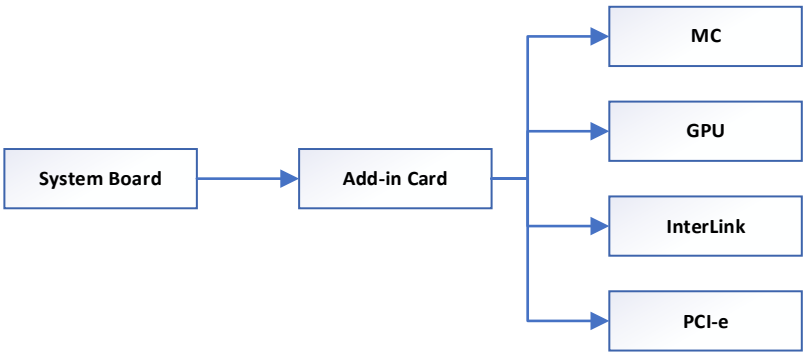

Figure 5 – Component Hierarchy

In the Figure below, the PDRs on the left side and the right side represent the same component hierarchy except via a single PDR vs multiple PDRs; either implementation is valid.

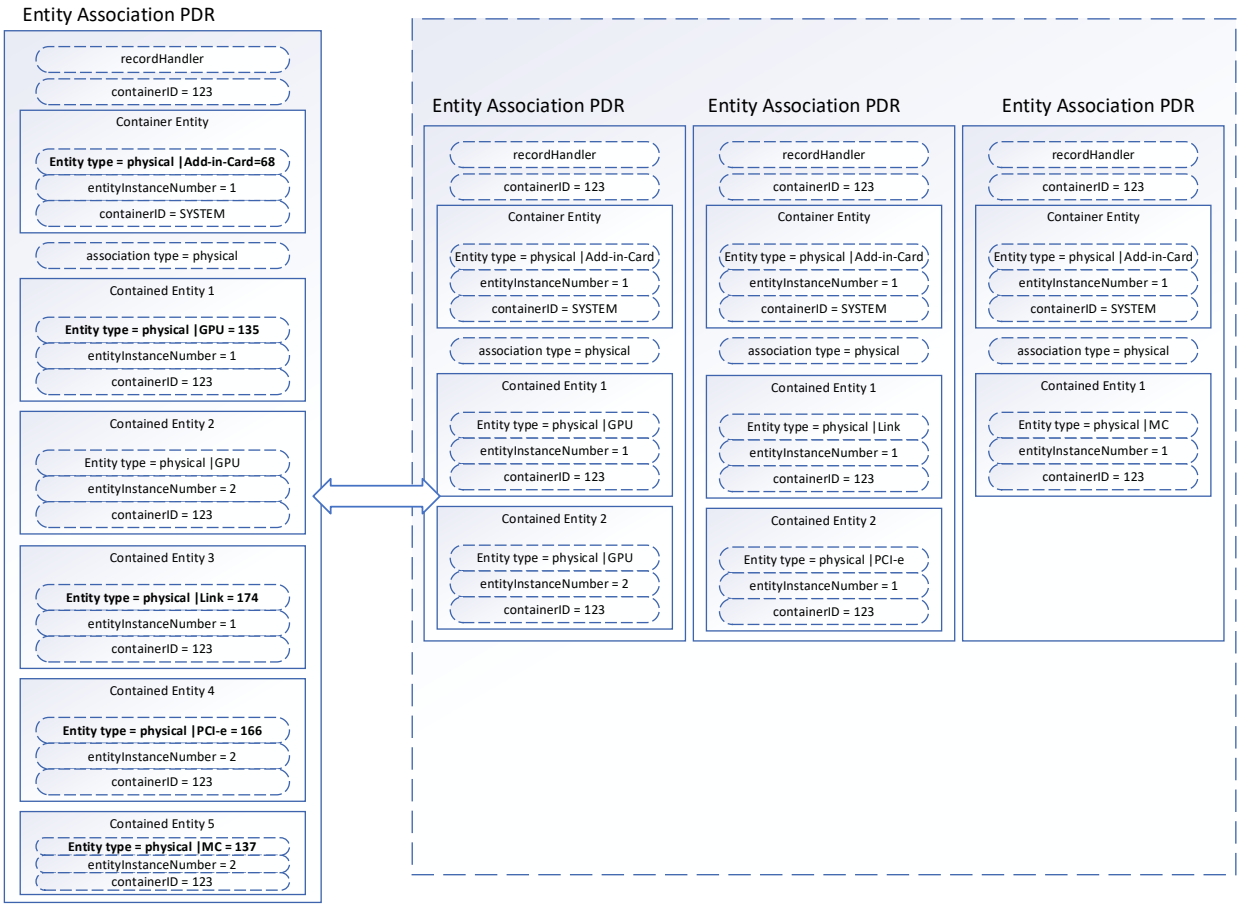

Figure 6 Entity Association PDRs

3.5.3.2 Logical Entity Association

Logical PDRs are used to associate a set of physical components to a logical entity. In this example the logical entity is “cooling domain”. This helps to group a set of sensors (in this example temperature sensors) of various physical modules to a single logical entity. This will help the management controller to associate these groups of sensors to the thermal algorithms used for fan and/or power control without requiring prior knowledge.

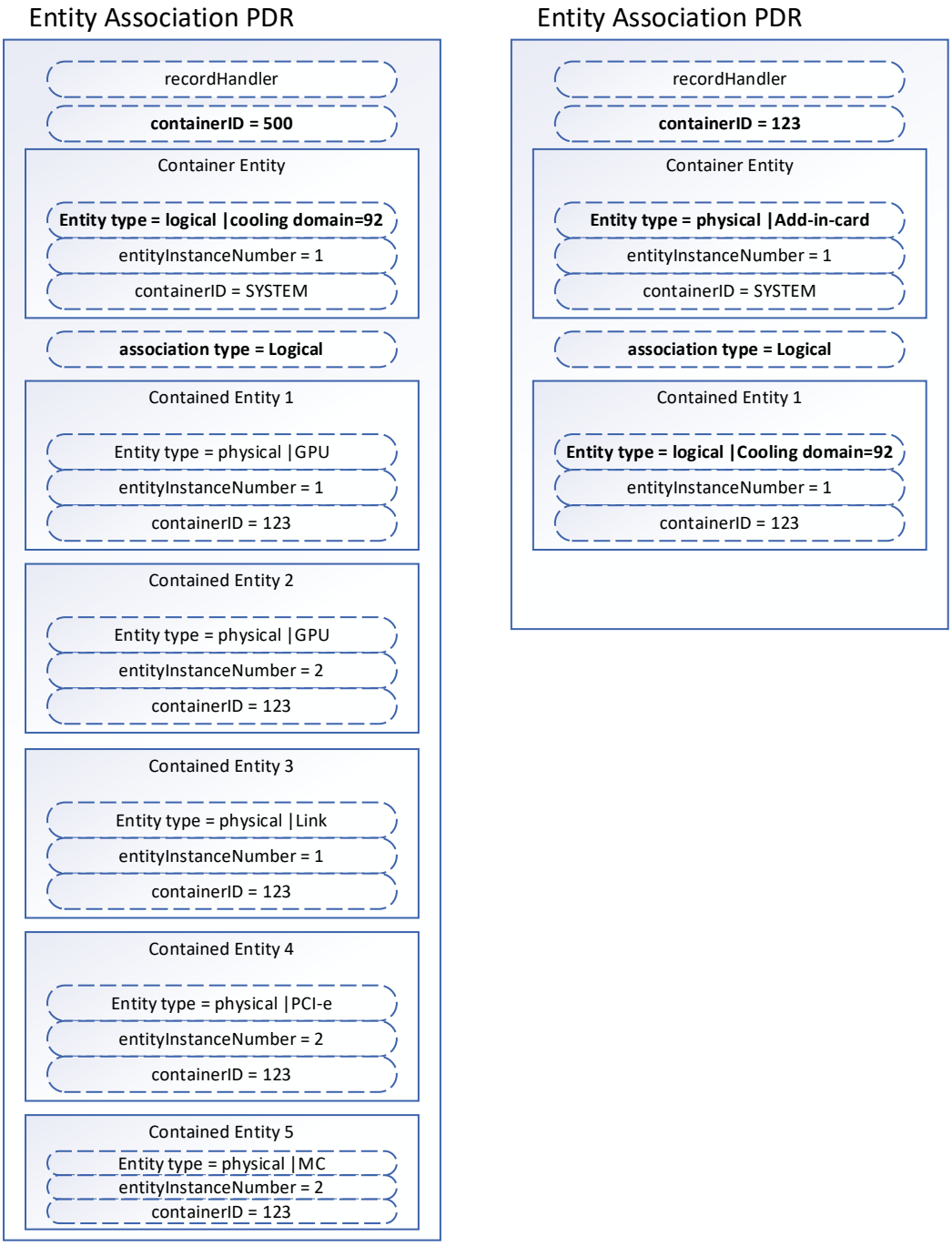

Figure 7 Logical Entity Association

3.5.4 Sensor & Effector Entity Mapping

The following section provides a list of examples showing how to map various sensors and effector to entity association PDRs so that the Host BMC can map the sensors to the actual physical modules.

3.5.4.1 Numeric sensors

In the following example, a temperature sensor with sensor ID 10 belongs to GPU 1 while sensor ID 20 belongs to an InterLink module.

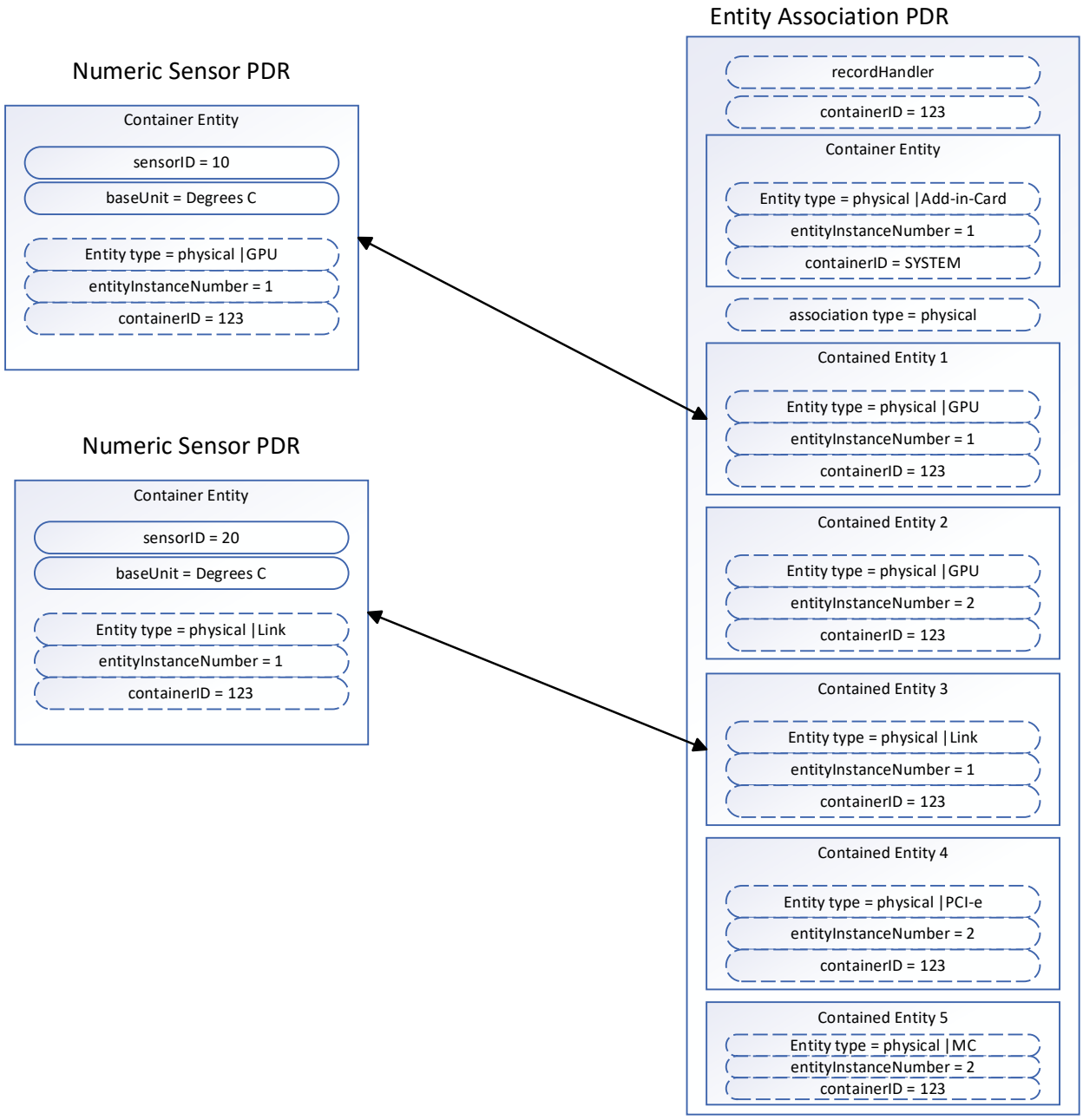

Figure 8 Numeric Sensor PDRs

3.5.4.2 State sensors

In this example, the Health state sensor is associated to an Add-in card to represent the overall health.

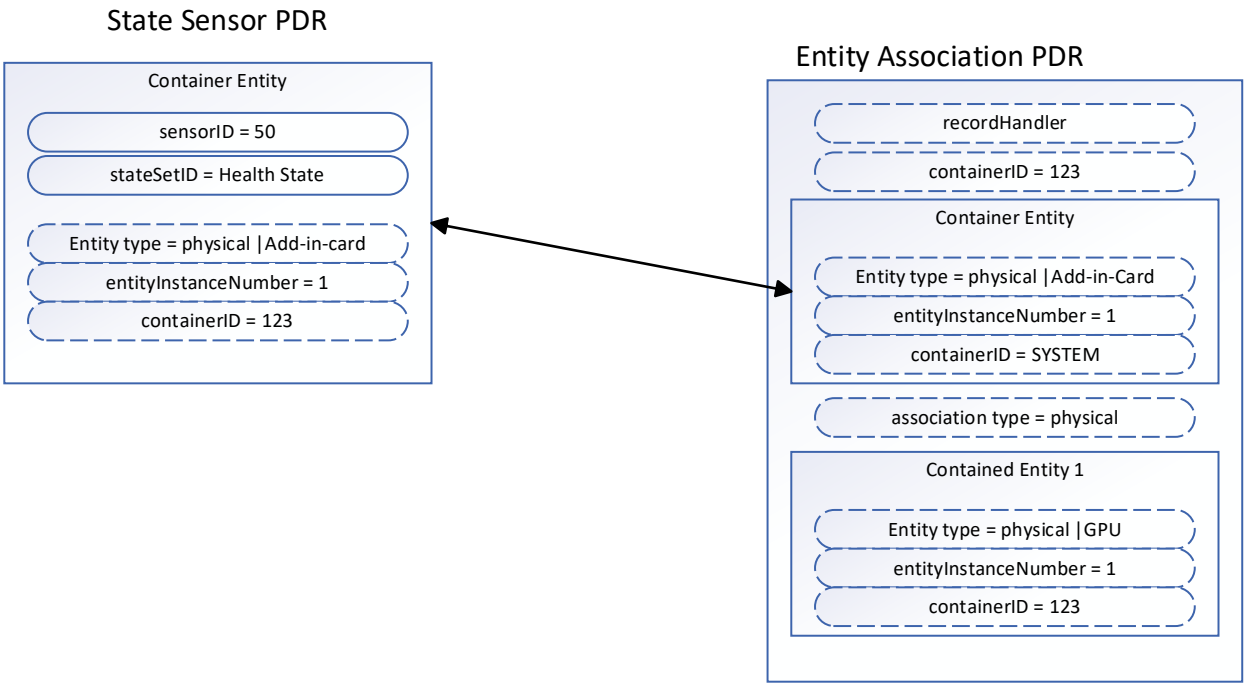

Figure 9 State Sensor PRD

3.5.4.3 Effectors

In this example, the numeric effector with ID 50 represents the overall power cap at the Add-in card level.

Likewise, the State Effectors “Reset” and “Reset to defaults” are associated to the management controller.

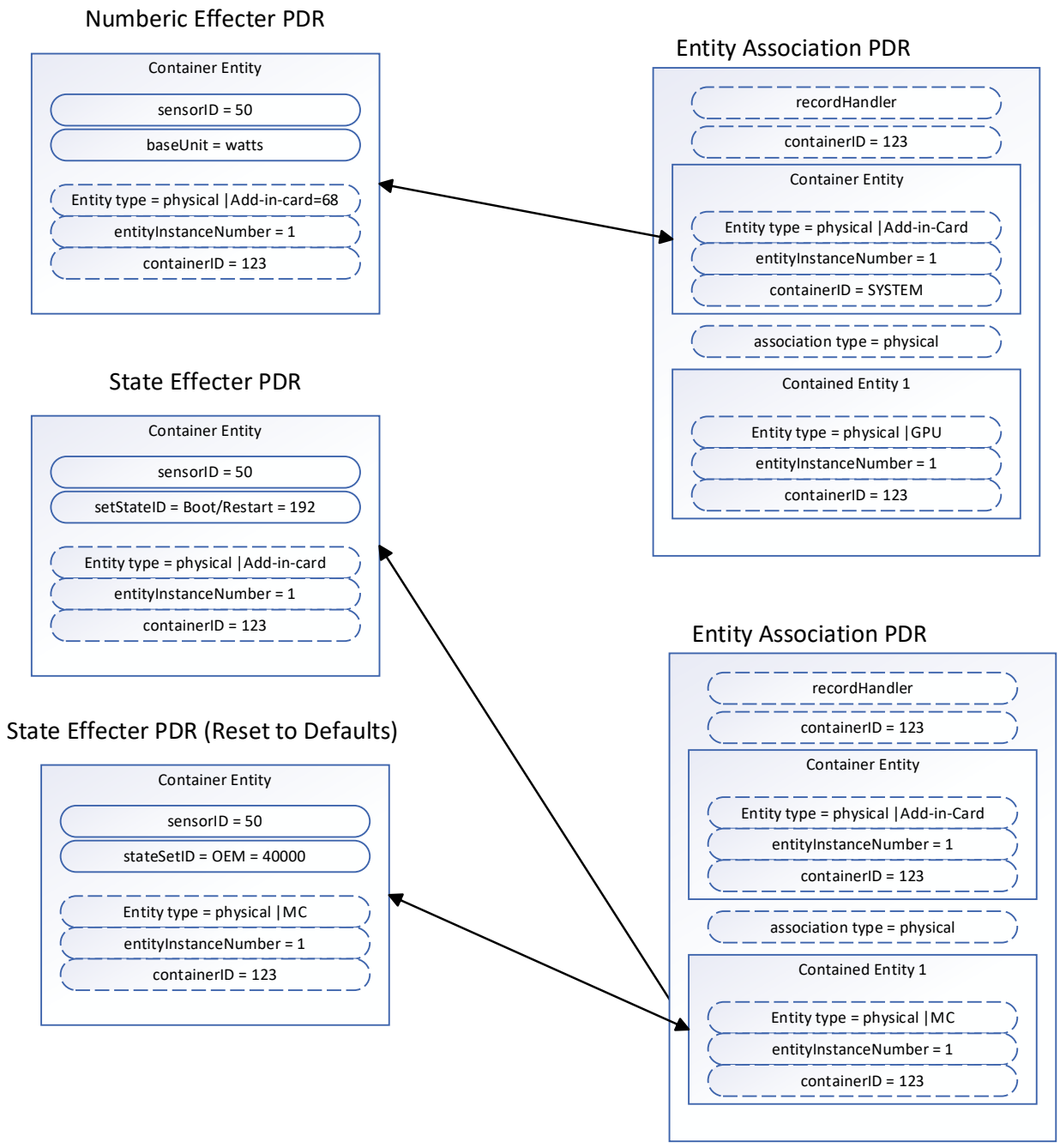

Figure 10 Effecter PDR

3.5.4.4 Logical sensor mapping

The mapping below associate sensor ID 15 of GPU 1 and sensor ID 25 of the InterLink module to the cooling domain entity.

Figure 11 Logical Sensor Mapping PDRs

3.5.5 PLDM Type IDs

The following table lists the Type IDs used to represent the Accelerator adapter as per DSP0249.

Table 11 – PLDM IDs

Component	Entity	Entity ID
Management Controller	Management controller	137
Add-in card	Add-in card	68
GPU	Processor	135
InterLink	Inter-processor bus	174
PCI-e	PCI Express Bus	166

3.6 MCTP OEM VDM

This section lists a reference implementation for how a device may expand telemetry data over MCTP when existing PLDM protocols are insufficient. It should be expected that device vendors will have additional OEM telemetry that will be harvested by the management controller that doesn't meet the existing definition of the PLDM Type 2 ecosystem. Fundamentally, the OCP group desires to push PLDM DMTF standards as quickly as possible but acknowledges that an OEM path for passing data is unavoidable.

3.6.1 Management Protocol Structures

3.6.1.1 Request Message Format

Table 12 – Request Message

	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bytes 0-3	Reserved				Hdr Version				Destination endpoint ID								Source endpoint ID								SOM	EOM	Pkt Seq #			TO	Message Tag	
Bytes 4-7	IC	Message type							PCI Vendor ID MSB								PCI Vendor ID LSB								RQ	D	Rsvd		Instance ID			
Bytes 8 - 11	OCP Message Type/Version								Vendor Message Type MSB								Vendor Message Type LSB								Data Size							
Bytes 12-15	Payload																															

3.6.1.2 Field Layouts

Table 13 – Request Message Field Layouts

Byte Offset	Bit Ranges	Contents
0x00	7-4	Reserved
	3-0	Header Version
0x01	7-0	Destination Endpoint ID
0x02	7-0	Source Endpoint ID
0x03	7	SOM
	6	EOM
	5-4	Packet Sequence Number

	3	TO
	2-0	Message eTag
0x04	7	IC
	6-0	Message Type = 0x7E
0x05 – 0x06	F – 0	PCI Vendor ID
0x07	7	RQ
	6	D
	5	Reserved
	4-0	Instance ID
0x08	7 – 4	OCP Type ID = 0x0C
	3 – 0	OCP Version = 0x01
0x09 – 0x0A	F – 0	Vendor Message Type
0x0B	8 – 0	Message Length, > 0
0x0C - <variable>		Message Data

3.6.1.3 Field Descriptions

Table 14 – Request Message Field Descriptions

Field Code	Field Size	Description
Message Type	7 bits	PCI vendor-defined message type. Always 0x7E.
PCI	2 bytes	PCI defined vendor ID
RQ	1 bits	Request bit. This bit is used to differentiate between request and other kind of messages. would be set to 1b for request messages and event messages. would be set to 0b for response messages.
D	1 bits	Datagram bit. This bit is used to indicate whether the instance ID is being used for asynchronous notifications (events) or for request/response tracking. Would be set to 1b for asynchronous notifications (events). Would be set to 0b for requests/responses. D and RQ bit combinations: 1. D = 0, RQ = 0b - Response message 2. D = 0, RQ = 1b - Request message 3. D = 1, RQ = 0b - Reserved 4. D = 1, RQ = 1b - Event messages
Rsvd	1 bits	Reserved.

Instance ID	5 bits	Used to uniquely identify a new instance of the API request. This is used to differentiate requests sent to the same MCTP endpoint and to match a particular instance of a request with a corresponding instance of the response.
OCP Designator	1 byte	7-4: 0x0C: OCP Designator 3-0: 0x01: OCP version
OEM Message Type	2 bytes	Vendor message type.
Data size	1 byte	Size of the trailing command-specific data in bytes. For requests which do not require an additional data payload this field must be 0.
Payload	variable	Optional command-specific data.

3.6.1.4 Response Message Format

Table 15 – Response Message

	Byte 1								Byte 2								Byte 3								Byte 4							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bytes 0-3	Reserved				Hdr Version				Destination endpoint ID								Source endpoint ID								SOM	EOM	Pkt Seq #			TO	Message Tag	
Bytes 4-7	IC	Message type							PCI Vendor ID MSB								PCI Vendor ID LSB								RQ	D	Rsvd		Instance ID			
Bytes 8 - 11	OCP Message Type/Version								Vendor Message Type MSB								Vendor Message Type LSB								Reason Code				Completion Code			
Bytes 12-15	Payload																															

3.6.1.5 Fields Descriptions

Table 14 – Response Message Fields Description

Field Code	Field Size	Description
TO	1 bit	Tag owner. Identifies whether the message tag was originated by the requester or the responder. Set to 0b for response messages.

RQ	1 bit	Request bit. This bit is used to differentiate between request and other kind of messages. Would be set to 1b for request messages and event messages. Would be set to 0b for response messages.
D	1 bit	Datagram bit. This bit is used to indicate whether the instance ID is being used for asynchronous notifications (events) or for request/response tracking. Would be set to 1b for asynchronous notifications (events). Would be set to 0b for requests/responses. D and RQ bit combinations: 1. D = 0, RQ = 0b - Request message 2. D = 0, RQ = 1b - Response message 3. D = 1, RQ = 0b - Reserved 4. D = 1, RQ = 1b - Event messages
Rsvd	1 bit	Reserved.
Instance ID	5 bits	Used to uniquely identify a new instance of the API request. This is used to differentiate requests sent to the same MCTP endpoint and to match a particular instance of a request with a corresponding instance of the response.
OCP Designator		7-4: 0x0C: OCP Designator 3-0: 0x01: OCP version
OEM Message Type	2 byte	OEM Message type. Represents a namespace for a group of related messages.
Reason code	3 bit	Used to return the command-specific reason code. This field expands on the status returned in the completion code field.
Completion code	5 bits	Used to return the status of the requested operation.
Data size	2 bytes	Data size.
Payload	variable	Command-specific response payload.

3.6.2 Completion Codes

Table 16 – Completion Codes

Value	Name	Description
0x00	SUCCESS	The command was accepted and completed successfully.
0x01	ERR_REQUEST	Generic error with processing the request.
0x02	ERR_INVALID_MSG_LENGTH	The request message length was invalid.
0x03	ERR_INVALID_MSG_TYPE	The message type used in the request is invalid or not supported by the responder.
0x04	ERR_INVALID_CMD_CODE	Invalid command code.

0x05	ERR_INVALID_ARG1	Invalid argument 1.
0x06	ERR_INVALID_ARG2	Invalid argument 2.
0x07	ERR_INVALID_DATA	The message payload contained invalid data or illegal parameter value.
0x08	ERR_CMD_NOT_SUPPORTED	The command code is not supported for this message type.
0x09	ERR_BUSY	The responder is unable to service the request at this time.
0x80 - 0xFF	Command Specific	Command specific error codes.

3.6.3 OEM Events

A future version of this document will describe OEM event extensions supporting OEM VDM telemetry for cases where standard PLDM events are insufficient/limited.

4 UBB Accelerator Device Interfaces

This section lists the minimum set of protocols and OEM extension/formats/schemas that any GPU/accelerator vendor needs to support on their UBBMC so that hyperscalers can seamlessly manage these devices without any additional vendor/device specific work.

4.1 Redfish

This section describes the high-level overview of Redfish organization. This section also proposes various Redfish resource behaviors and OEM schema extensions.

4.1.1 Redfish Tree

The Redfish tree diagram offers a comprehensive overview of the UBBMC management capabilities and resource organization. It provides an overview and relationships between different components. The resource label “UBB” represents the complete Accelerator subsystem/enclosure within the server and the resource label “OAM_X” represents each of the Accelerator module assembly.

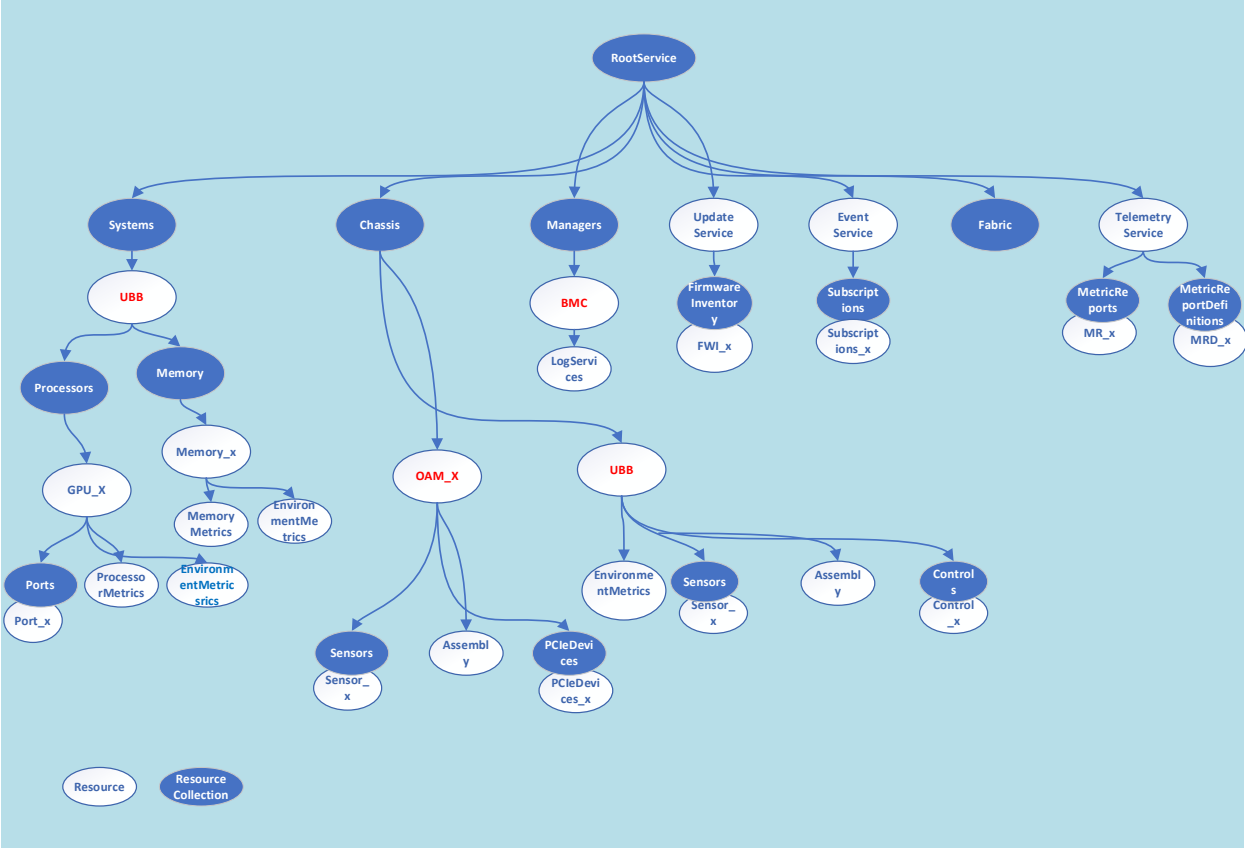

Figure 12 UBB Redfish Model

4.1.2 Location Objects

The Location object referenced from many Redfish resources is very important in enabling automation at scale. CSPs use various techniques to correlate the contents of the Location object into unambiguous mappings to physical hardware in their data centers.

Table 17 – Location Object Example

```
"Location":
{
  "PartLocationContext": "Backplane 3, Slot 4",
  "PartLocation":
  {
    "LocationType": "Port",
     "ServiceLabel": "Port 1"
  }
}
```

Location property example.

CSPs often rely on a tuple formed by the combination of the `Location.PartLocationContext`, and `Location.PartLocation` to bind entities to external logical management controls. For this reason, the Location resource, and its `PartLocationContext`, `PartLocation.LocationType`, and `PartLocation.ServiceLabel` fields are required by CSPs in any resource that supports `Location`.

4.1.3 RAS Error Injection

The following section provides details about the error injection requirements for various hardware component levels. This information is described in greater detail in the OCP GPU & Accelerator RAS Requirements v0.5 specification. This document provides specific Redfish resource examples for Memory and PCIe. These examples can be applied to other resources that GPU/accelerator vendors can support as OEM extensions. The errors are injected via Redfish actions.

4.1.3.1 Injectable Hardware Components

The current version of this document provides schema details for two hardware components, memory and the PCI-e interconnect.

Table 18 – Error Types

Error type	Details	Priority/Comments
Memory	GPU SRAM and GPU DRAM (HBM)	DDR memory is part of Host Motherboard and is excluded from here.
PCIe errors	• PCIe Switches, PCIe Network devices, PCIe End point Devices • PCIe Re-timers • PCIe Link	PCIe re-timers require special attention as their errors are different from general PCIe endpoint devices and PCIe Switches. PCIe Links will have Link Width, Link Speed, and Link Down Error considerations.

4.1.3.2 Memory Error Injection Attributes

Table 19 – Memory Error Attributes

Attribute Type	Details	Comments
Physical Device Identification	Uses Redfish based URL	Specifies the physical or logical device that is the target of error injection.
Sub Device Identification	Rank, Column, Row Level	Additional details to pinpoint a specific physical or logical device that is the target of the error injection. Optional
Error Severity	Correctable Uncorrectable	Each device will have a specific enumeration of allowed errors to inject. Those are specified here.

Address	Memory Address location where to Inject Error	Further refined location of the memory address that should reflect the injected error. Optional

4.1.3.2.1 Memory Error Injection Details

The latest Redfish schema (V_1_17_0) added a new action property "InjectPersistentPoison" in support of memory error injection. This document defines two additional OEM OCP actions, InjectCorrectableError and InjectUncorrectableError. The table below provides details of these actions. Sample Redfish is provided later in this document.

Table 20 – Memory Errors

Property Name	Schema(s)	Parameters	Type	Description
InjectPersistentPoison	Memory(Actions)	PhysicalAddress,	Object	Injects poison to the memory address in the memory device.
InjectCorrectableError	OcpMemory(Actions)	PhysicalAddress	Object	Injects correctable errors to the memory address in the memory device.
InjectUncorrectableError	OcpMemory(Actions)	PhysicalAddress	Object	Injects an uncorrectable error to the memory address in the memory device.

Note: A future version of this document is expected to include additional memory error injection attributes related to device physical location (e.g., row, column, bank, and rank.)

InjectPersistentPoison property defined in Memory schema.

Table 21 – PersistentPoison Properties

```
"InjectPersistentPoison": {
  "description": "Injects poison to a persistent memory address in the
memory device.",
    "parameters": {
      "PhysicalAddress": {
         "description": "The device physical address as a hex-encoded
string.",
           "requiredParameter": true,
           "type": "string"
         }
      },
   "type": "object",
   }
}
```

4.1.3.2.2 InjectCorrectableError & InjectUncorrectableError

Table 22 – Properties

```
 "InjectCorrectableError": {
   "description": "Injects a correctable error to a specific persistent
memory address in the memory device. ",
     "parameters": {
       "PhysicalAddress": {
           "description": "The device physical address as a hex-encoded
string.",
            "requiredParameter": true,
            "type": "string"
    }
  },
},
"InjectUncorrectableError": {
  "description": "Injects an uncorrectable error to a specific persistent
memory address in the memory device. ",
  "parameters": {
      "PhysicalAddress": {
          "description": "The device physical address as a hex-encoded
string.",
           "requiredParameter": true,
           "type": "string"
    }
  }
}
```

4.1.3.2.3 Actions

This section details the actions that can be performed on memory components.

Table 23 – Memory Action Properties

```
"@odata.id": "<Memory ResourceUri>”,
"Actions": {
  "Oem": {
    "OCP":{
      "#OcpMemory.InjectCorrectableErrors": {
        "target": "<Memory
ResourceUri>/Actions/Oem/OcpMemory.InjectCorrectableErrors",
          "@Redfish.ActionInfo": "<Memory
ResourceUri>/Oem/OCP/InjectCorrectableErrorActionInfo"
      },
      "#OcpMemory.InjectUncorrectableErrors": {
        "target": "<Memory
ResourceUri>/Actions/Oem/OcpMemory.InjectUncorrectableErrors",
        "@Redfish.ActionInfo": "<Memory
ResourceUri>/Oem/OCP/InjectUncorrectableErrorActionInfo"
      }
    }
  }
```

```
}
```

4.1.3.3 PCIe Errors Injection Attributes

Table 24 – PCIe Errors

Attributes	Details	Comments
Device Identification	Uses Redfish based URL	Specifies the physical or logical device that is the target of error injection.
Error Severity	PCIe Correctable PCIe Non-Fatal and PCIe Fatal	Each device will have a specific enumeration of allowed errors to inject. Those are specified here.
Error Type	Correctable e.g., Bad TLP, Bad DLLP, Receiver Error, Reply Timeout Ref [1] Non Fatal e.g., Poisoned TLP received, Completion Timeout, Unexpected Completion Ref [1] Fatal e.g., Malformed TLP, Flow control Protocol Error, Training Error, Receiver Overflow. Ref [1]	Each device will have a specific enumeration of allowed errors types to inject. Those are specified here.

4.1.3.3.1 Properties

Table 25 – PCIe Error Properties

```
  ...
"Oem": {
      "OCP": { [
            " InjectCorrectableError ": {
                  "additionalProperties": false,
                  "description": "Injects Correctable Error to a specific
PCIe device.",
                   "parameters": {
                         "ErrorType": {
                                "enum": [
                         "ReceiverError",
                                   "BadTLP",
                                   "BadDLLP",
                                   "ReplayTimerTimeout",
                                   "ReplayNumRollover"
                               ],
                                    "type":"string"
                                    }
                               }
                   }
            "InjectUncorrectableNonFatalError": {
                  "additionalProperties": false,
                  "description": "Injects Uncorrectable NonFatalError to a
specific PCIe device.",
```

```
                    "parameters": {
                          "ErrorType": {
                                  "enum": [
                          "PoisonedTLPReceived",
                                    "ECRCCheckFailed",
                                    "UnsupportedRequest",
                                    "CompletionTimeout",
                                    "CompleterAbort",
                                    "UnexpectedCompletion"
                               ],
                                    "type":"string"
                                    }
                               }
                     }
            "InjectUncorrectablFatalError": {
                  "additionalProperties": false,
                  "description": "Injects UncorrectableFatalError to a
specific PCIe device.",
                    "parameters": {
                          "ErrorType": {
                                  "enum": [
                          "TrainingError",
                                    "DLLProtocolError",
                                    "ReceiverOverflow",
                                    "FlowControlProtocolError",
                                    "MalformedTLP"
                               ],
                                    "type":"string"
                                    }
                               }
                     }
                  ]
           }
}
```

4.1.3.3.2 *Actions*

Table 25 – PCI-e Error Actions

```
  "@odata.id":"<PCIeDevice ResourceUri>",
   "Actions": {
     "Oem":{
       "OCP":{
         "#OcpPCIeDevice.InjectCorrectableError": {
           "target": "<PCIeDevice
ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectCorrectableError",
           "@Redfish.ActionInfo": "<PCIeDevice
ResourceUri>/Oem/OCP/InjectCorrectableErrorActionInfo"
         },
         "#OcpPCIeDevice.InjectUncorrectableNonFatalError": {
           "target": "<PCIeDevice
ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectUncorrectableNonFatalError",
           "@Redfish.ActionInfo": "<PCIeDevice
ResourceUri>/Oem/OCP/InjectUncorrectableNonFatalErrorActionInfo"
         },
```

```
        "#OcpPCIeDevice.InjectUncorrectableFatalError": {
          "target": "<PCIeDevice
ResourceUri>/Actions/Oem/OcpPCIeDevice.InjectUncorrectableFatalError",
          "@Redfish.ActionInfo": "<PCIeDevice
ResourceUri>/Oem/OCP/InjectUncorrectableFatalErrorActionInfo"
        }
      }
    }
```

4.2 Out-of-Band Interfaces (MCTP, PLDM, and SPDM)

UBB designs are connected to the enclosing system via high-speed networking, where Redfish is the interface, and out-of-band connections including I2C/I3C, where MCTP, PLDM, and SPDM are the interface protocols.

The implementation of the out-of-band protocols is same as with discrete GPU/Accelerator devices described in the above sections except that it out-of-band communication is limited to specific purposes such as monitoring high-frequency telemetry for thermal and power related features.

5 Conclusion

The management profiles for GPUs in this document provide guidance for how GPU devices can be designed and how CSPs can manage them. These profiles reduce the amount of work required for GPU suppliers to bring new products to market by converging CSP requirements around industry standards. Likewise, these profiles reduce the effort and time required for CSPs to bring new GPU designs to market by enabling a higher level of code and test re-use between different GPU parts and vendors.

6 Glossary

BMC – Baseboard Management Controller. This is a management controller present within system designs that frequently interacts with the management of discrete accelerator devices and UBBs.

CSP – Cloud Service Provider

Hyperscaler – Cloud Service Provider

UBB – Universal baseboard; a type of accelerator delivered as a large board containing multiple GPUs and high-speed fabric interconnect.

UBBMC – the Universal Base Board Management Controller. This is the Redfish controlled management controller present on an OCP OAI HGX UBB board.

7 References

- OCP GPU & Accelerator RAS Requirements v0.5
- OCP Attestation of System Components

- Platform Management FRU Information Storage Definition rev. 1.3 (IPMI)
- DMTF DSP0236 revision 1.3.1 or later (MCTP)
- DMTF DSP0274 revision 1.2.1 or later (SPDM)
- DMTF DSP0240 revision 1.1.0 or later (PLDM Type 0)
- DMTF DSP0248 revision 1.2.2 or later (PLDM Type 2)
- DMTF DSP0267 revision 1.2.0 or later (PLDM Type 5)
- DMTF DSP0266 revision 1.18.0 or later (Redfish)

8 License - Open Web Foundation (OWF) CLA

Contributions to this Specification are made under the terms and conditions set forth in Open Web Foundation Modified Contributor License Agreement ("OWF CLA 1.0") ("Contribution License") by:

Google, Microsoft, NVIDIA

Usage of this Specification is governed by the terms and conditions set forth in **Open Web Foundation Modified Final Specification Agreement ("OWFa 1.0.2") ("Specification License").**

You can review the applicable OWFa1.0 Specification License(s) referenced above by the contributors to this Specification on the OCP website at http://www.opencompute.org/participate/legal-documents/. For actual executed copies of either agreement, please contact OCP directly.

Notes:

1. The above license does not apply to the Appendix or Appendices. The information in the Appendix or Appendices is for reference only and non-normative in nature.

NOTWITHSTANDING THE FOREGOING LICENSES, THIS SPECIFICATION IS PROVIDED BY OCP "AS IS" AND OCP EXPRESSLY DISCLAIMS ANY WARRANTIES (EXPRESS, IMPLIED, OR OTHERWISE), INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, OR TITLE, RELATED TO THE

SPECIFICATION. NOTICE IS HEREBY GIVEN, THAT OTHER RIGHTS NOT GRANTED AS SET FORTH ABOVE, INCLUDING WITHOUT LIMITATION, RIGHTS OF THIRD PARTIES WHO DID NOT EXECUTE THE ABOVE LICENSES, MAY BE IMPLICATED BY THE IMPLEMENTATION OF OR COMPLIANCE WITH THIS SPECIFICATION. OCP IS NOT RESPONSIBLE FOR IDENTIFYING RIGHTS FOR WHICH A LICENSE MAY BE REQUIRED IN ORDER TO IMPLEMENT THIS SPECIFICATION. THE ENTIRE RISK AS TO IMPLEMENTING OR OTHERWISE USING THE SPECIFICATION IS ASSUMED BY YOU. IN NO EVENT WILL OCP BE LIABLE TO YOU FOR ANY MONETARY DAMAGES WITH RESPECT TO ANY CLAIMS RELATED TO, OR ARISING OUT OF YOUR USE OF THIS SPECIFICATION, INCLUDING BUT NOT LIMITED TO ANY LIABILITY FOR LOST PROFITS OR ANY CONSEQUENTIAL, INCIDENTAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND EVEN IF OCP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

9 OCP Tenets

Openness

- This specification was developed via close and open collaboration between industry partners and competitors.
- All specifications and interfaces produced through this effort will be available to all OCP members.

Efficiency

- The goal of this specification is to make integration of GPUs into Hypercaler solutions seamless, reducing toil for both the supplier and the hyperscaler consumers.
- A companion to this effort is an OCP compliance tool that will enable automated validation of these interfaces, ensuring reduced toil and high-quality products

Impact

- This document represents the first industry initiative to standardize GPU requirements between suppliers and hyperscale consumers.
- The advances in this document are expected to have significant impact on quality and time-to-market for GPU systems deployed by hyperscalers.
    - These advances will also be applicable and beneficial to enterprise deployments of GPU systems.

Scale

- This specification applies to very large-scale GPU system deployments in hyperscaler data centers.

Sustainability

- The profiles defined in this document enable cross-generational commonality for key functionality of GPU parts, enabling logistics to support longer lifespan of GPU parts and a healthy secondary market for these parts.

10 About Open Compute Foundation

At the core of the Open Compute Project (OCP) is its Community of hyperscale data center operators, joined by telecom and colocation providers and enterprise IT users, working with vendors to develop open innovations that, when embedded in product are deployed from the cloud to the edge. The OCP Foundation is responsible for fostering and serving the OCP Community to meet the market and shape the future, taking hyperscale led innovations to everyone. Meeting the market is accomplished through open designs and best practices, and with data center facility and IT equipment embedding OCP Community-developed innovations for efficiency, at-scale operations and sustainability. Shaping the future includes investing in strategic initiatives that prepare the IT ecosystem for major changes, such as AI & ML, optics, advanced cooling techniques, and composable silicon. Learn more at www.opencompute.org.