Core Information Model (CoreModel)

TR-512.A.9 Appendix – Processing Construct Examples

Version 1.5
September 2021

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

Disclaimer

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

Important note

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

Table of Contents

List of Figures

Document History

Version	Date	Description of Change
		Appendix material was not published prior to Version 1.3
1.3	September 2017	Version 1.3 [Published via wiki only]
1.3.1	January 2018	Addition of text related to approval status.
1.4	November 2018	No change.
1.5	September 2021	Enhancements to model structure

1 Introduction to the document suite

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

1.1 References

For a full list of references see TR-512.1.

1.2 Definitions

For a full list of definition see TR-512.1.

1.3 Conventions

See TR-512.1 for an explanation of:

- UML conventions
- Lifecycle Stereotypes
- Diagram symbol set

1.4 Viewing UML diagrams

Some of the UML diagrams are very dense. To view them either zoom (sometimes to 400%) 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.

1.6 Appendix Overview

This document is part of the Appendix to TR-512. An overview of the Appendix is provided in TR-512.A.1.

2 Introduction to this Appendix document

This document provides various examples of the use of the ProcessingConstruct model to represent complex functions.

The examples in this document extend the simple examples given in TR-512.11.

3 General examples

3.1 Types of Processing Construct

In this section, we will go through a number of different types of 'device' and show how the ProcessingConstruct concept allows us to represent them all in a consistent way.

3.1.1 Traditional 'Device'[1]

The simplest common case that we have is where we have a physical unit that is logically a single unit and is managed as a single unit. That is, the physical, logical and management scopes are congruent.

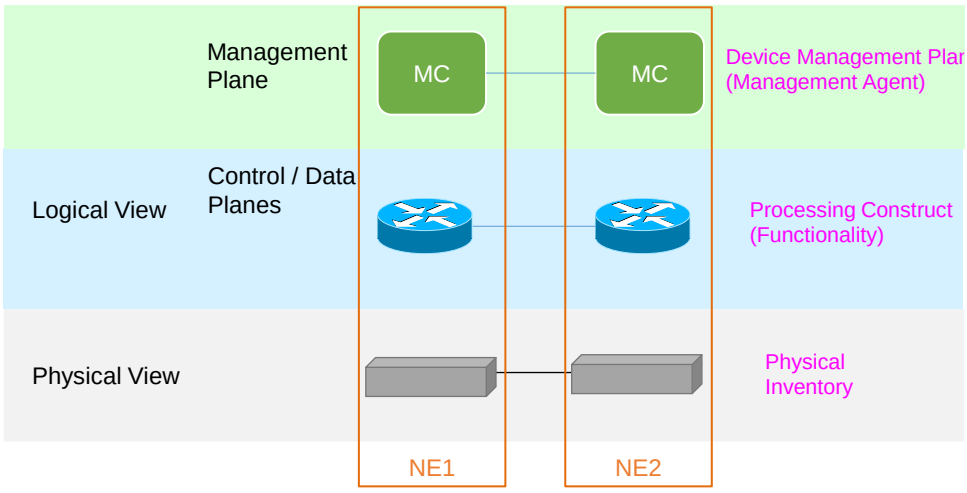


Figure 3-1 Traditional 'Device' representation deconstructed

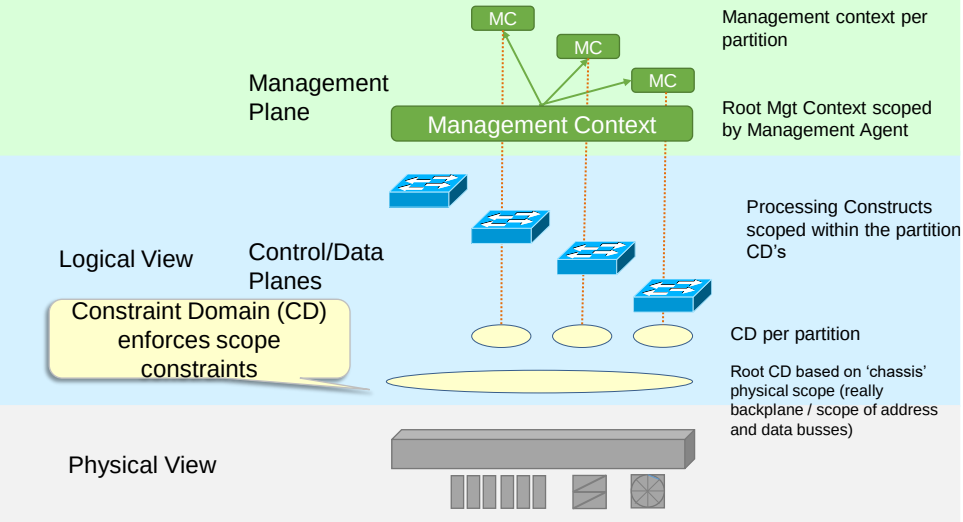
Because a number of simple devices fit this special case, unfortunately it was used as the general case, which is problematic for the other cases that we will now look at.

[1] Here we will use the term ‘device’ in a loose and undefined manner to aid in the discussion. The term is not defined because it is not important for our discussion, the generally understood concept is sufficient

3.1.2 Partitioned 'Device'

For this example, we will assume that we have a single physical unit that can be logically partitioned and that the management is also partitioned. There may be a number of variations on this theme but we will just cover this basic case.

The traditional NetworkElement concept can't effectively represent this case because it assumes congruence between the physical, logical and management scopes.



The management plane may be global or partitioned, or both (as shown).
Root MC, Root CD and Physical Inventory have same scope.

Figure 3-2 Partitioned 'Device'

We will now look at how to use the ProcessingConstruct and ConstraintDomain classes to model this case.

The first thing to understand is that there will be some constraints related to the physical scope, and a ConstraintDomain instance should be created to support that.

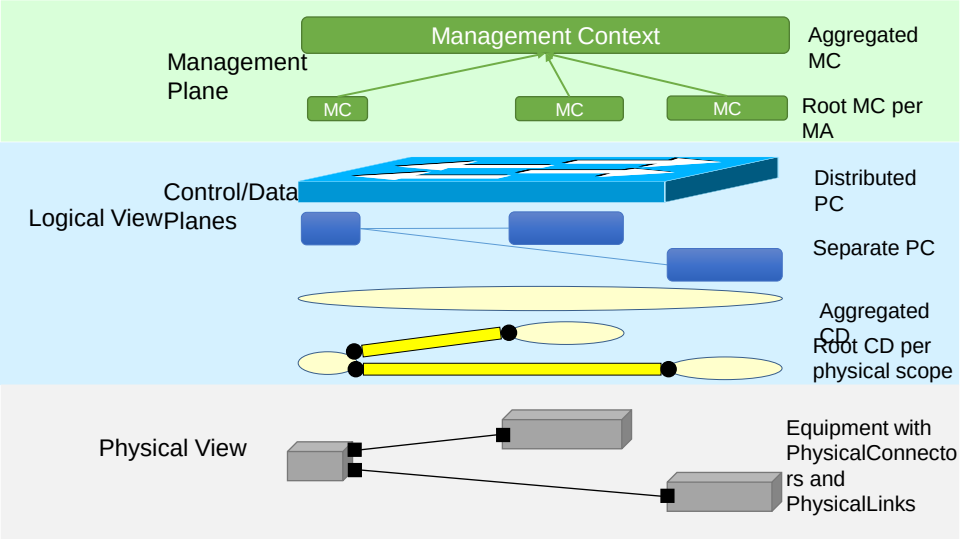
Secondly, we also have constraints that are related to the partitions, and a ConstraintDomain should be created per partition.

Note that in our example, one of the domains has a larger scope than the others – this will depend on how the partitioning works and there may be many options – the important thing is to create the CD to match the scopes.

3.1.3 Distributed 'Device'

For this example, we will assume that we have many physical units that can be logically aggregated to behave as a single logical unit and that the management is also aggregated. There may be a number of variations on this theme but we will just cover the basic case where we have one central unit and one or more 'satellite' units.

Again, the traditional NetworkElement concept can't effectively represent this case because it assumes congruence between the physical, logical and management scopes.



The management plane may be global or partitioned, or both (as shown).
Root MC, Root CD and Physical Inventory have same scope.

Figure 3-3 Distributed 'Device' with separate Management Agents

We will now look at how to use the ProcessingConstruct and ConstraintDomain classes to model this case.

The first thing to understand is that there will be some constraints related to the physical scopes, and ConstraintDomain instances should be created to support that.

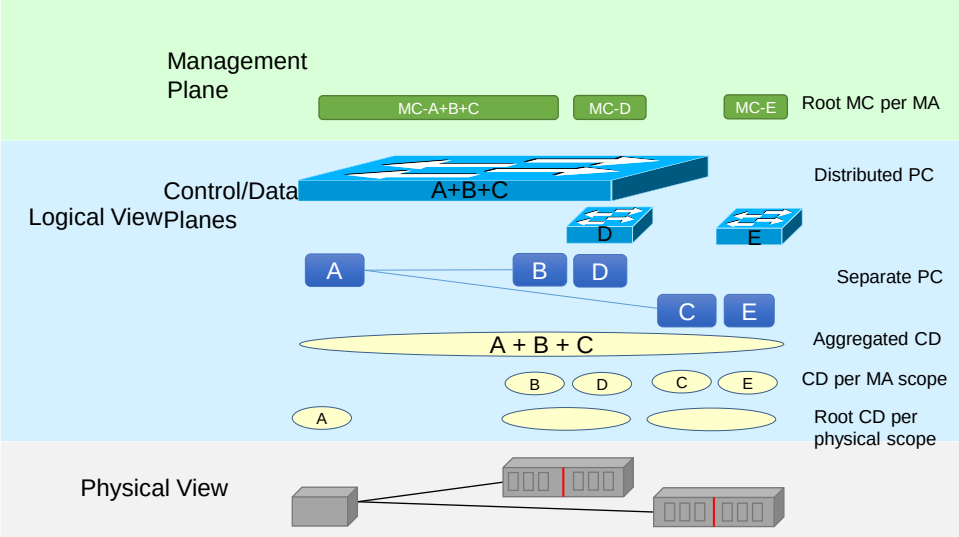
Secondly, we also have constraints that are related to the aggregated functionality, and a ConstraintDomain should be created for the 'distributed device'.

ProcessingConstructs should be created to map to how the functionality works; some PC may be constrained by the physical scopes and some may span the entire logical device. Partial control plane synchronization technologies[2] like ICCP (Inter-Chassis Communication Protocol RFC 7275), vPC (Virtual Port Channel, multi-chassis link aggregation) would be associated with the 'distributed device' Constraint Domain. In these cases, there are multiple control planes (where only part of the configuration is synchronized).

In the management plane, there may be some independent device management, or the management may be only at the distributed device level. Lightweight Wireless Access Points that are remotely managed may be an example of the aggregated management case. Aggregating technologies like stacked switches and VSS that merge complete control planes are another example of this case (two peer devices share the one control plane and the one MA).

[2] http://www.cse.wustl.edu/~jain/cse570-13/ Multi-Tenant Isolation and Network Virtualization in Cloud Data Centers, slide 4

Shown below is a more complex example where we combine partitioning and aggregation. If you try and draw a 'NE boundary' similar to those in the figure above, you will quickly find that there isn't a sensible answer.



The management plane may be global or partitioned, or both (as shown).
Root MC, Root CD and Physical Inventory have same scope.

Figure 3-4 Logically Split Chassis

The steps to model this are the same as before:

- Create ConstraintDomains to represent the actual constraint scopes
- Create ProcessingConstructs and assign them to the relevant ConstraintDomains

3.1.4 'Virtual Device'

Here we will assume that we have a physical host (of some form factor) that is running a virtualization technology (Virtual Machine or Container) that is running software that provides some managed functionality.

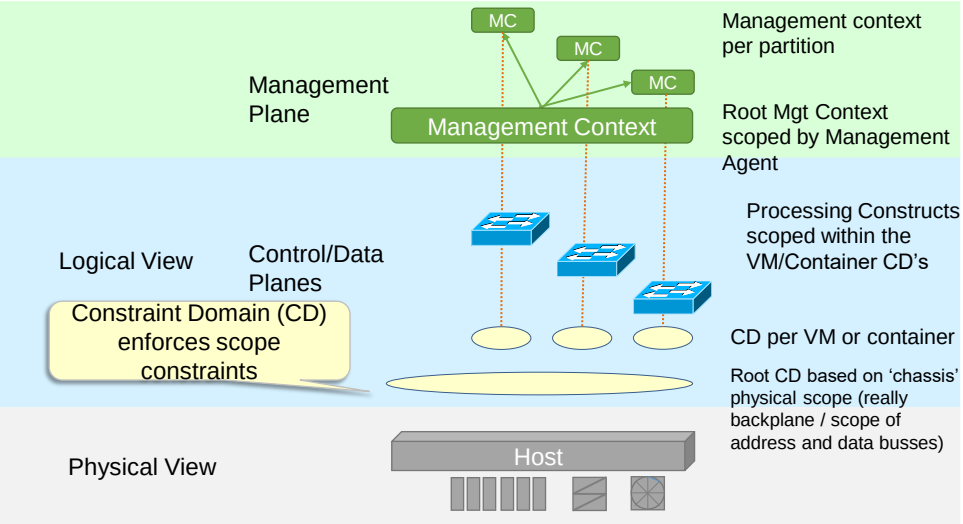


Figure 3-5 "Virtual" Device

We will now look at how to use the ProcessingConstruct and ConstraintDomain classes to model this case.

The first thing to understand is that there will be some constraints related to the physical scope, and a ConstraintDomain instance should be created to support that.

Secondly, we also have constraints that are related to each VM / Container, and a ConstraintDomain should be created for each of these. Note that in this release of the ONF CIM, we don't have a software model that would allow us to represent the guest and host operating systems or the hypervisor / VMM (Virtual Machine Manager).

3.1.5 'Virtual Distributed Device'

For this example, we will assume that we have many 'virtual' units that can be logically aggregated to behave as a single logical unit and that the management is also aggregated. There may be a number of variations on this theme but we will just cover the basic case where we have one central unit and one or more 'satellite' units.

As shown in the diagram below, we have a distributed virtual switch (DVS), where a VM runs a central switch module and remote modules run on other VMs. Note that other complete or partial control plane synchronization technologies[3] like stacked switches and VSS, ICCP (Interchassis Communication Protocol RFC 7275), vPC (Virtual Port Channel, multi-chassis link aggregation) would be associated with a 'distributed device' Constraint Domain.

[3] http://www.cse.wustl.edu/~jain/cse570-13/ Multi-Tenant Isolation and Network Virtualization in Cloud Data Centers, slide 4

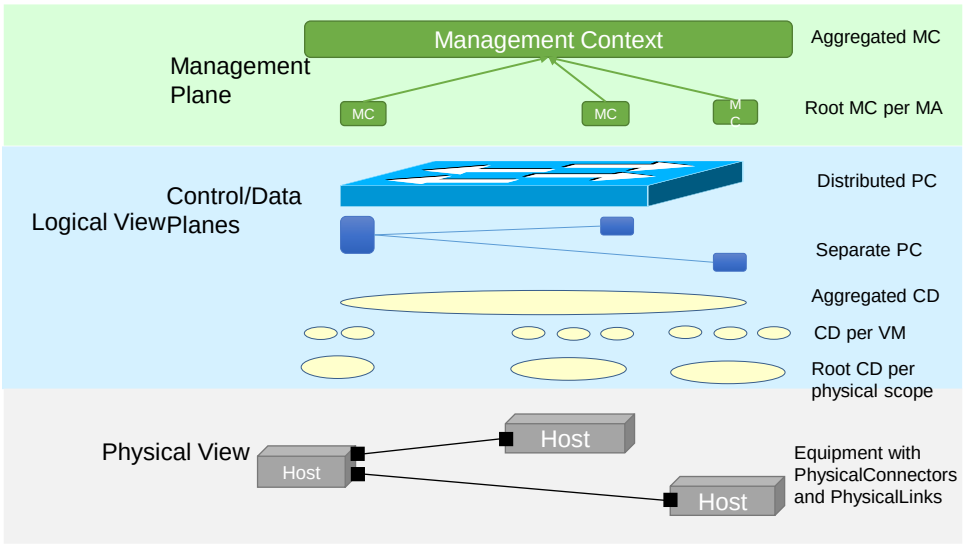


Figure 3-6 "Virtual" Distributed Device

We will now look at how to use the ProcessingConstruct and ConstraintDomain classes to model this case.

The first thing to understand is that there will be some constraints related to the physical host scopes, and ConstraintDomain instances should be created to support that.

Secondly, we also have constraints that are related to each VM / Container, and a ConstraintDomain should be created for each of these. Note that in this release of the ONF CIM, we don't have a software model that would allow us to represent the guest and host operating systems or the hypervisor / VMM.

Thirdly we also have constraints that are related to the aggregated functionality, and a ConstraintDomain should be created for the 'distributed device'.

ProcessingConstructs should be created to map to how the functionality works; some PC may be constrained by the physical scopes and some may span the entire logical device.

In the management plane there may be some independent device management, or the management may be only at the distributed device level.

3.1.6 SDN Controller[4]

Shown below is a simplified block diagram of a SDN controller.

The SDN controller itself is similar to (the control plane of) a network element, so we will represent it using a ConstraintDomain. PeerContext is a generalization of client and server

[4] Deeper examples that show the relationship to the general control model will be added in a later release.

context and because it is a scoping concept, representing it as a ConstraintDomain would be appropriate.

If other scoping concepts such as resource groups are required, then they would also be ConstraintDomains.

Inside the SDN controller will be a number of processing constructs. Similar to our other examples this could include things like a BGP routing control process, a PTP clock control process or an ERP G.8032 control process. The SDN controller itself isn't one massive ProcessingConstruct.

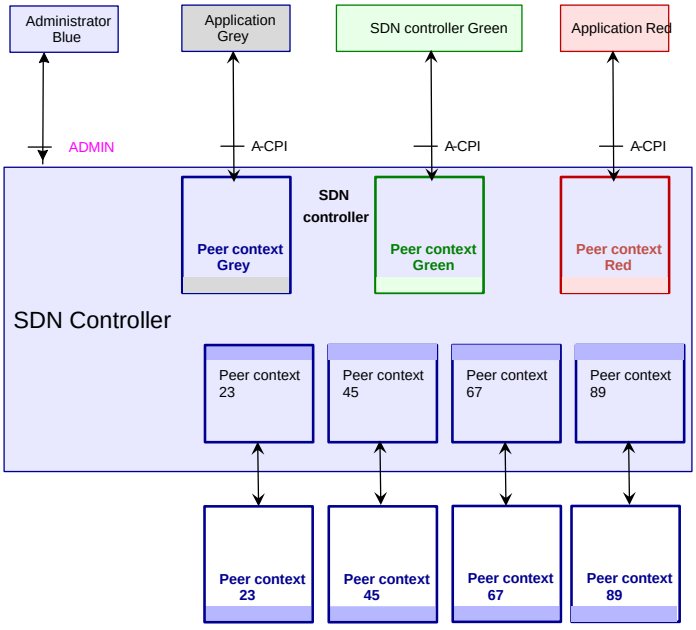


Figure 3-7 SDN Controller

The SDN controller host (physical or VM container) would be represented as shown in the other examples.

Also, the PC/CD model can cope with the SDN controller being centralized or distributed, as shown in the other examples.

3.1.7 Other 'Devices'

The previous examples show a number of basic ways that functionality can be related to the underlying hardware and to the management plane.

There may be other real world cases (that are probably hybrids of these fundamental cases), but the important thing is that the decoupled model design allows for many other options.

3.2 PTP Clock Example

The figure below is covered at a high level in TR-512.11 and now we will look at it in more detail.

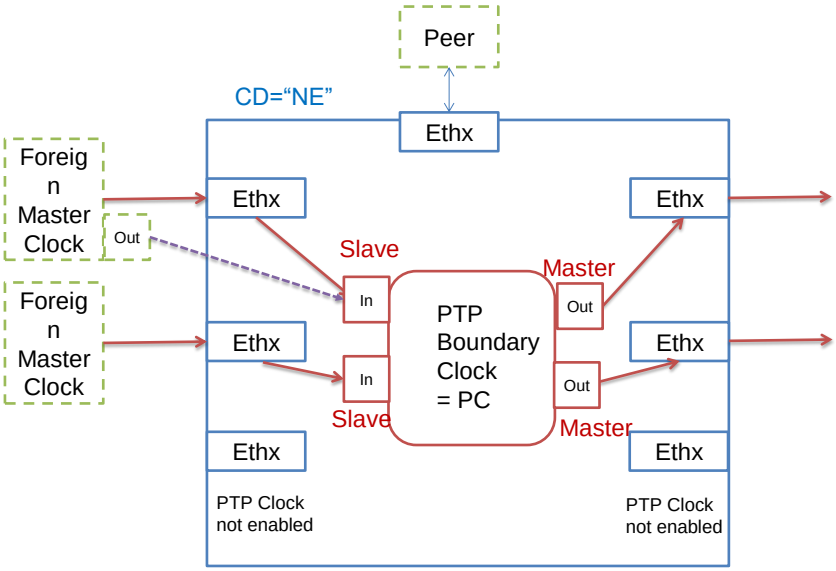


Figure 3-8 PTP Clock Concepts

Given the functional block diagram above, the question is what the resultant model should be.

The UML class diagram below shows a possible solution.

The PTP clock PC function (in the figure above) is in PtpClockFunction class (in the figure below) and the attributes of the PcPort (shown as "In" and "Out" in the figure above) are in PtpClockPort (in the figure below).

The PTP protocol also has a domain concept, where clock domains form separate topologies (clocks only peer when the domain matches). A ConstraintDomain can be used to show this network level constraint (PtpClockDomain below).

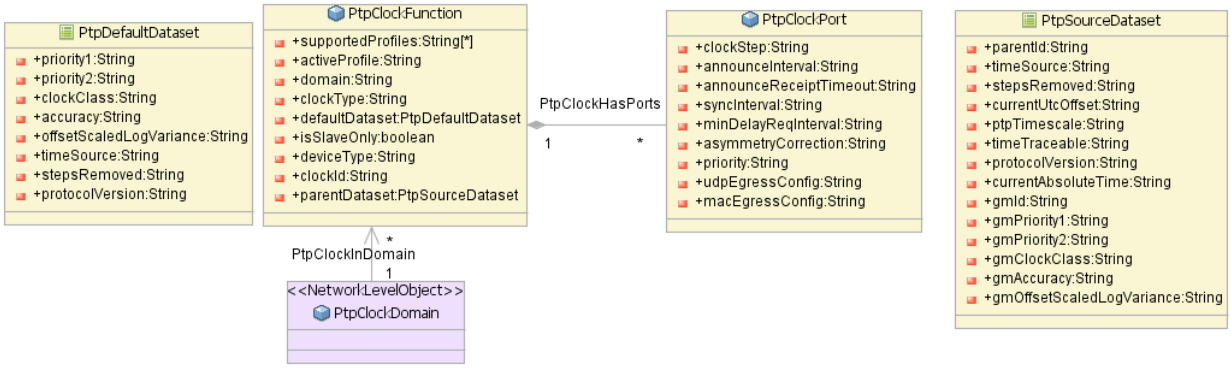


Figure 3-9 PTP Model sketch

3.3 ERPS G.8032 Example

The figure below is covered at a high level in TR-512.11, and now we will look at it in more detail.

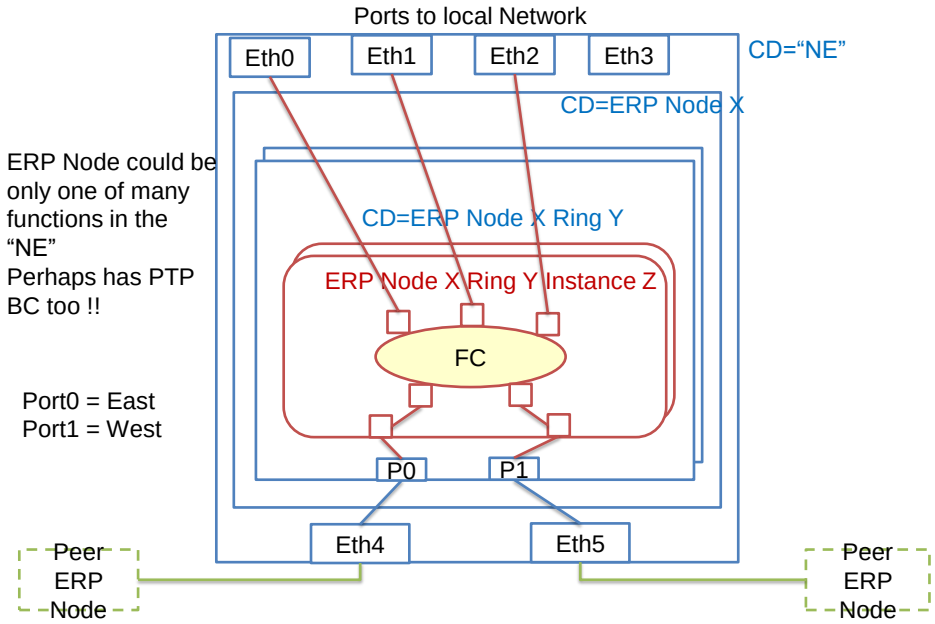


Figure 3-10 ERP G.8032 Concept Example

Given the functional block diagram above, the question is what the resultant model should be.

The UML class diagram below shows a possible solution.

ErpNodeCd is a ConstraintDomain that can be used to group all of the ERP rings in the device (shown as ERP Node X in the figure above). It can also be used to hold any 'device level ERP global attributes'.

ErpRingNode is a ConstraintDomain representing the part of the ring in the 'device' (shown as ERP Node X Ring Y in the figure above) and has the related Po and P1 port configuration attached.

ErpInstanceNode is the ProcessingConstruct representing the instance of the ring on the 'device' (shown as ERP Node X Ring Y Instance Z in the figure above)[5] and has its related port configurations.

An ErpRingNode can contain many ErpInstanceNodes.

As discussed in the main document, the network level scopes of ErpRing and ErpRingInstance can be represented using ConstraintDomain and the relevant PC related to these CD.

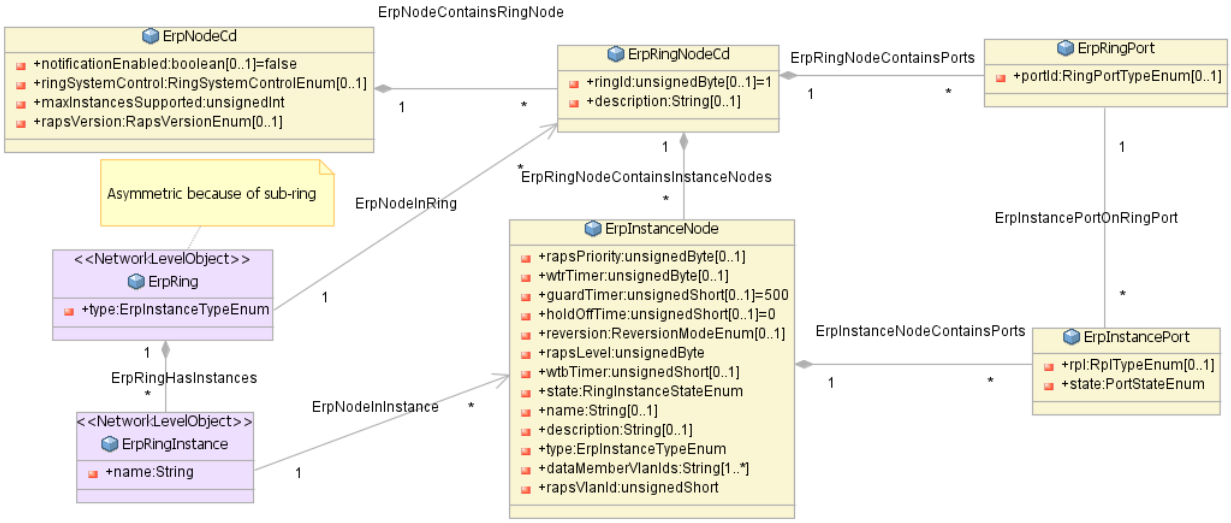


Figure 3-11 ERP Model Sketch

End of Document

[5] ErpInstance is a G.8032 term and shouldn't be confused with model class instances