Open Networking Foundation

Core Information Model (CoreModel)

TR-512.11

Processing Construct

Version 1.6
January 2024

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

Disclaimer

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

Important note

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

Table of Contents

List of Figures

Document History

Version	**Date**	**Description of Change**
		This document 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 changes.
1.5	September 2021	Enhancements to model structure.
1.6	January 2024	Updated release and dates.

1 Introduction to the document suite

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

1.1 References

For a full list of references see TR-512.1.

1.2 Definitions

For a full list of definition see TR-512.1.

1.3 Conventions

See TR-512.1 for an explanation of:

- UML conventions
- Lifecycle Stereotypes
- Diagram symbol set

1.4 Viewing UML diagrams

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

1.5 Understanding the figures

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

2 Introduction to Processing Construct

The ProcessingConstruct (PC) represents generalized functionality. The PC is used in conjunction with the ConstraintDomain (CD), that groups PCs and constrains their usage. In addition to being generaly applicable to represent functionality that is not being modeled in detail, the PC and CD form the fundamental pattern that allows an important transition in the representation of a 'device'[1].

The ONF Common Information Model represents a 'device' with a logical functionality viewpoint utilizing an underlying physical inventory viewpoint. This decoupling of logical and functional scopes is important for us to proceed to the next stage in our representation, the removal of NetworkElement[2].

NetworkElement (now deprecated) was the key concept that was used to represent logical functionality. In this version of the model, the new ProcessingConstruct (PC) and ConstraintDomain (CD) classes are used as a basis for the replacement of the NetworkElement concept (see TR-512.8).

This document describes the PC and CD and provides some detail of the model.

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

3 Purpose and essentials of Processing Construct

3.1 Background

In the ONF CIM there are already separate classes for special types of functions:

- ForwardingConstruct to represent forwarding functionality
- LogicalTerminationPoint to represent termination, and
- ForwardingDomain to represent forwarding scope constraints.

ProcessingConstruct is in addition to these concepts and is to be used where the major function of interest is related to processing rather than forwarding of information.

While there are a number of grey areas between processing and forwarding, there are a few 'pure' processing constructs:

- Memory
- CPU
- Storage

Another use for ProcessingConstruct is for representing control plane processes such as packet routing processes. Packet routers commonly run many routing protocols and may also run many

---

[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.

[2] Deprecation of the formal concept and of the corresponding model class.

instances of each routing protocol. Each routing process instance peers independently and using ProcessingConstruct we can show the actual control plane topologies.

While it's hard to give extensive rules where ProcessingConstruct should be used, probably the best advice is to use it where significant processing of information takes place. Also it may be useful to "Think of the network as composed of PC, constrained by CD and connected via FC".

ConstraintDomain has been introduced to add a 'container' concept. ConstraintDomain is similar to ForwardingDomain, but more general. Note that ConstraintDomain is designed as a lightweight scope boundary only, the functionality resides in PC, FC and LTP.

A useful way to think of the change is that the NetworkElement was like a black box that obscured what was happening inside. ConstraintDomain is like a light, clear wrapper around the functions, grouping and constraining them, but not adding or obscuring anything.

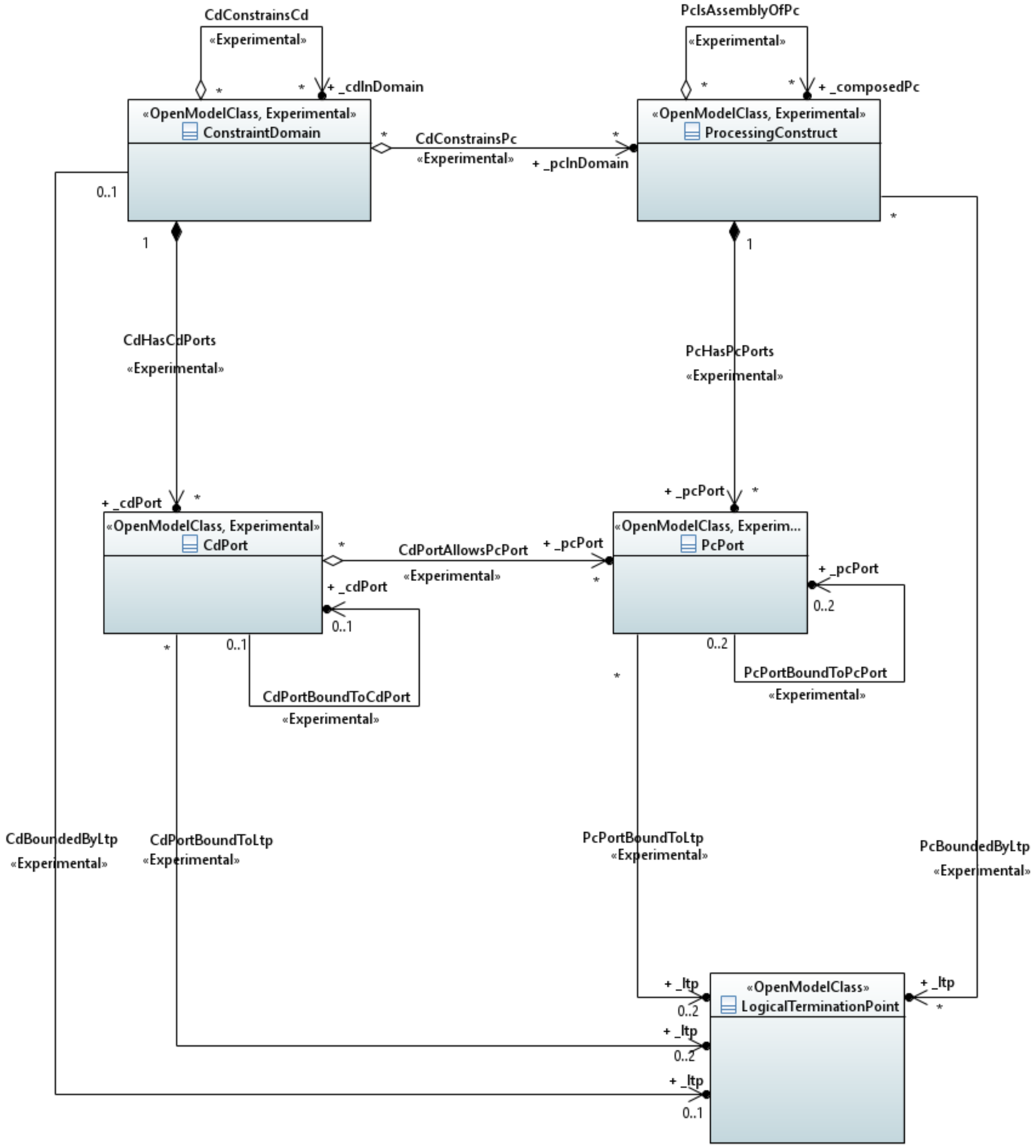
ConstraintDomain represents a general functional scope that can be used for grouping and scope boundaries. ConstraintDomains can be related, they can overlap and they don't need to form a hierarchy.

ProcessingConstruct is likely to be extremely useful in the "Application Layer" which tends to be more about processing and less about forwarding.

3.2 Model

In a number of places in the ONF CIM, the Component-Port pattern is used (e.g. ForwardingDomain & FdPort, ForwardingConstruct & FcPort). ConstraintDomain is a more generic version of ForwardingDomain and has a related CdPort class to provide at least equivalent capability. ProcessingConstruct is similar to ForwardingConstruct and has a related PcPort class to provide necessary capability for the many asymmetric cases.

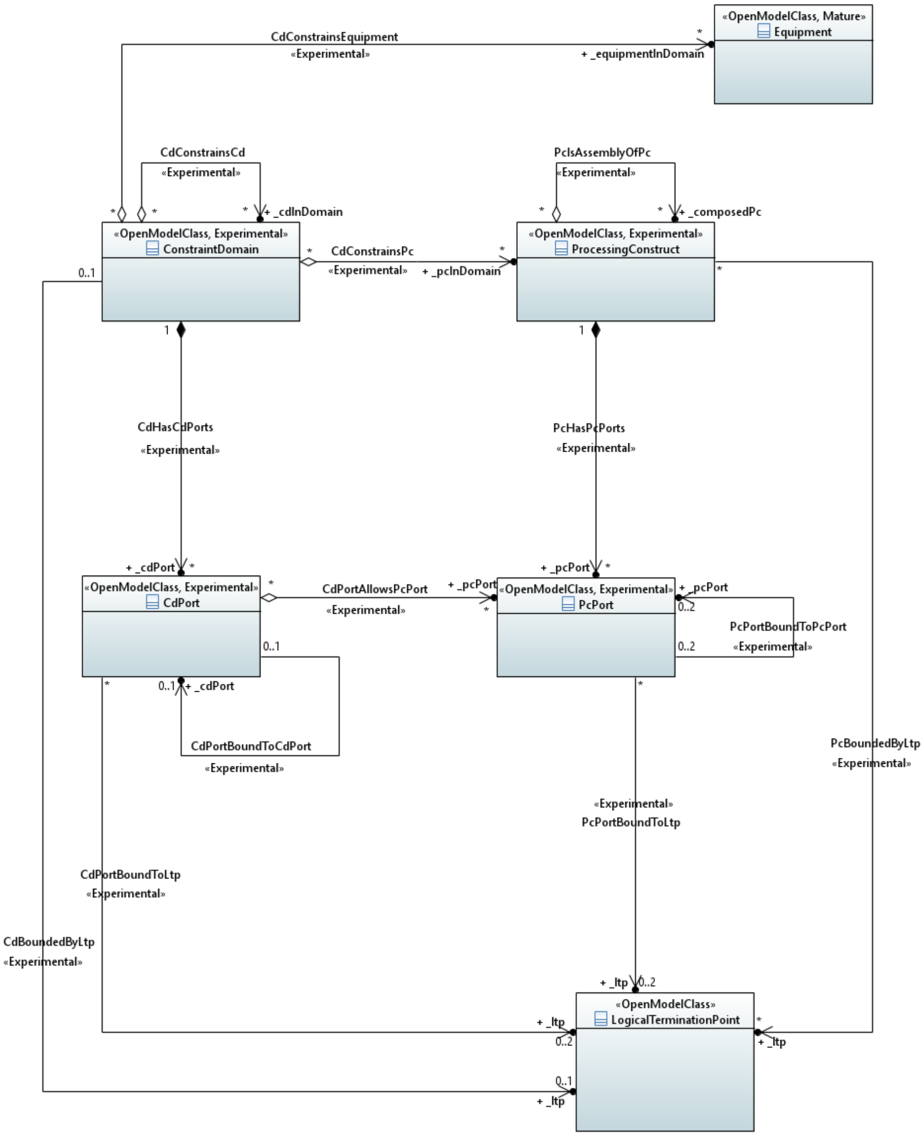
In summary, we have added ConstraintDomain and ProcessingConstruct to the ONF CIM using our standard patterns. The diagram below shows the core model and emphasizes the pattern usage.



CoreModel diagram: ProcessingConstruct-Core

Figure 3-1 Processing Construct and Constraint Domain core model

The figure below shows the relationship between the PC/CD model and the Equipment model. The figure highlights an enhancement in the relationship between equipment and the function model (covered in detail in TR-512.6). The model emphasizes that PC can represent anything from the most fundamental small (atomic) functions to the richest and most sophisticated of functions. The recursion shown allows the equipment to give rise to a block of functionality that can be decomposed into atomic parts and then those parts can be reassembled, perhaps through a number of intermediate forms to eventually give rise to LTPs etc. In addition, at any level of assembly, the functionality can be made resilient by selecting from two or more alternatives[3].



CoreModel diagram: ProcessingConstruct-EquipmentAndLtpWithCore

Figure 3-2 PC and CD with Equipment and LTP

In the figure below other model classes have been added. The figure emphasizes the similarity between the patterns of the CD/PC model and the FD/FC model. The FD/FC is essentially a narrow usage of CD/PC.

In the model (as shown in the figure below):

---

[3] Clearly this is a basic scheme and further work is required in this area. It is likely that a level of sophistication similar to the FC resilience will be required. In addition there is a need to model load sharing and other analogue relationships which has also not been covered.

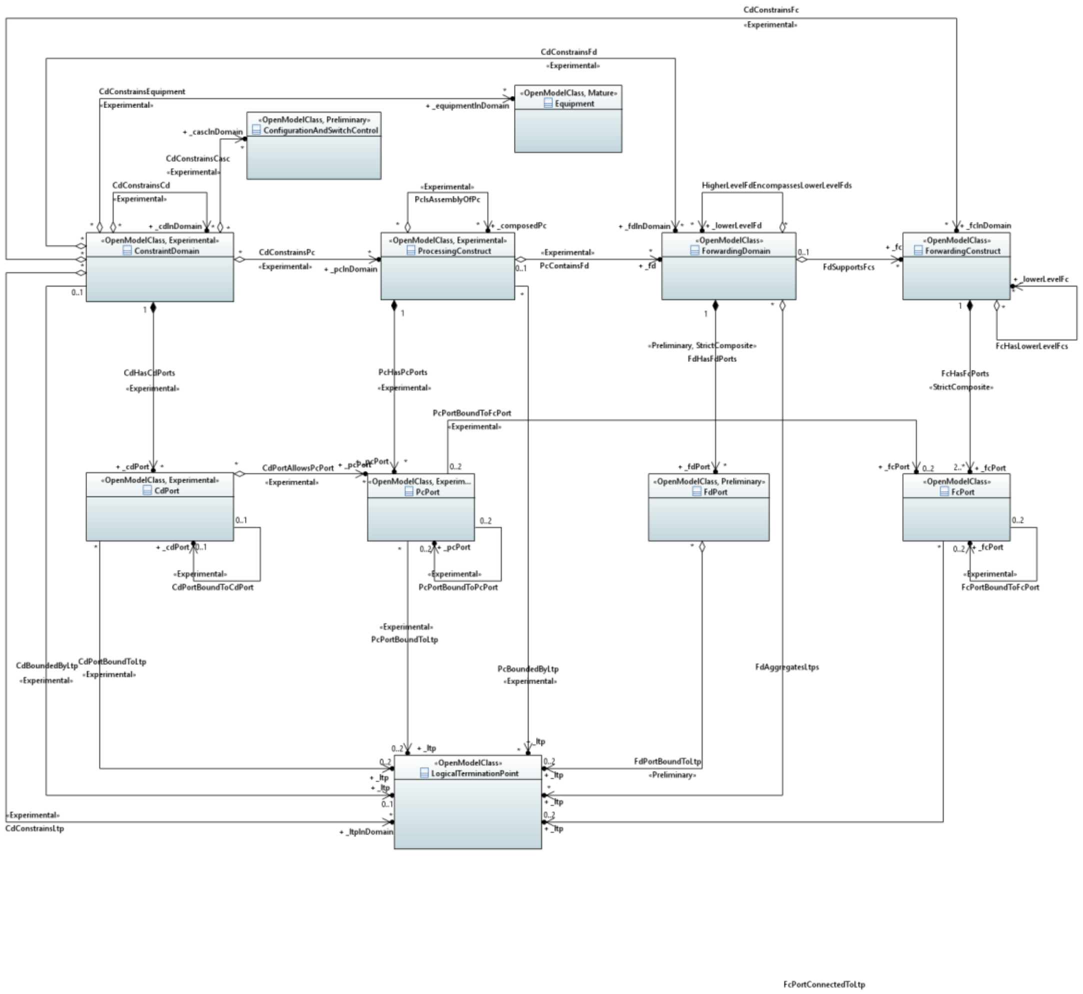
- One or more ConstraintDomains can constrain ProcessingConstruct via the CdConstrainsPc association (this is a more general capability than that of ForwardingDomain constrains ForwardingConstruct).
- Two or more ConstraintDomains can be related together via the CdEncompassesCd association (in the same way the ForwardingDomain can).
- One or more ConstraintDomains can constrain ForwardingConstruct via the CdConstrainsFc association.
- PcPort is related to LogicalTerminationPoint in the same way that FcPort is.
- ConstraintDomain can constrain ForwardingDomains (note that ConstraintDomain and ForwardingDomain are very similar concepts)[4]
- ConstraintDomain association to Equipment allows us to relate the constraint boundary to an underlying physical entity.
- ForwardingDomain can constrain ProcessingConstruct (in the same way that ForwardingDomain constrains ForwardingConstruct)[5].
- The association "EquipmentGivesRiseToFunctionalBlock" applies to a low level function that is wholly "contained" by an FRU[6].

---

[4] Because of the similarity of functionality between CD and FD it makes sense to review CD and FD in a future release development cycle to determine if there is an opportunity for convergence.

[5] This and the reverse association are highlighted in red as the rationale for the associations requires further work.

[6] There are two potential issues with this association: (1) navigation direction wrt coupling of model domains (2) essential duplication with PcRealized…association. There are some special subtle challenges highlighted here: (1) at this base level the navigation is strongly the reverse of the PcRealized… association (2) it is strongly compositional unlike PcRealized... This is a case where association navigability, strength and multiplicity vary per case. This area will be addressed in the next release.



CoreModel diagram: ProcessingConstruct-FullModel

Figure 3-3 PC and CD with FD, FC and C&SC

The sections below provide details of the classes and relationships.

Note that not all attributes/details are shown for the classes below (see TR-512.DD for a list of all attributes). Only those attributes that are relevant for this document are shown.

3.3 PC/CD model

3.3.1 ProcessingConstruct (PC)

Qualified Name: CoreModel::ProcessingConstructModel::ObjectClasses::ProcessingConstruct

ProcessingConstruct (PC) can be used to represent both potential and enabled processing between two or more of its PcPorts.
A PcPort may be associated with another PcPort or with an LTPs at a particular specific layerProtocol.
Like the LTP, the PC supports any transport protocol including all circuit and packet forms.
The PC is used to effect processing of information extracted from the transport layer protocol

signal.
A PC may be:
- Asymmetric such that it offers several functions and such that different functions are offered to different attached entities.
- Symmetric such that it offers (or is considered as offering) only one function and the same function is offered to any attached entity with no interactions between functions offered to each attached entity
An asymmetric PC offering a number of distinct functions will have PcPorts through which the distinct functions are accessed.
A symmetric PC offering only a single function need not have PcPorts, the function can be accessed directly from the PC.

Inherits properties from:

- GlobalClass

This class is Experimental.

Table 1: Attributes for ProcessingConstruct

Attribute Name	**Lifecycle Stereotype (empty = Mature)**	**Description**
_pcPort	Experimental	An asymmetric PC instance is related to LTPs via PcPorts (essentially the ports of the PC). Symmetric PCs don't have PcPorts and are directly related to LTPs.
_ltp	Experimental	A symmetric PC instance is associated with zero or more LTP objects. The LTPs on the PC boundary provide information for processing and capacity for communication. For asymmetric PCs the association to the LTP is via the PcPort (with stated role, allowing access to a specific function of the PC).
_fd	Experimental	One or more ForwardingDomains can constrain a ProcessingConstruct. A constrained PC connects LTPs on the boundary of the FD.
_composedPc	Experimental	The PC class supports a recursive aggregation relationship (PcIsAssemblyOfPc). This allows both: - abstraction where an assembly of PCs (forming a System) is viewed as an abstract PC - decomposition such that the internal construction of a PC can be exposed as multiple lower level PCs. Appropriate use of this association allows each of a collection of PCs to be decomposed into atomic parts (PCs) that can then be assembled into set of complex functions where each function in the set can be viewed as a PC. Note that the model actually represents an aggregation of lower level PCs into higher level PCs as viewpoints rather than partitions, and supports multiple views.
_pcResilienceSelector	Obsolete	PcResilienceSelector that realizes the resilience of the PC.
_runningSoftwareProcess	Experimental	See referenced class
_runningEquipment	Experimental	See referenced class

3.3.2 PcPort

Qualified Name: CoreModel::ProcessingConstructModel::ObjectClasses::PcPort

Represents the access to the functionality of a PC.

Inherits properties from:

- LocalClass

This class is Experimental.

Table 2: Attributes for PcPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
portRole		Identifies the role of the port in the context of the specification of the PC.
_ltp	Experimental	A PC instance is associated with zero or more LTP objects. The LTPs on the PC boundary provide capacity for processing. For asymmetric PCs the association to the LTP is via the PcPort.
_pcPort	Experimental	A PcPort can be directly bound to another PcPort (rather than via a LTP) to support a simplified application level view (rather than requiring the full transport level view).
_fcPort	Experimental	A PcPort can be directly bound to an FcPort (rather than via a LTP) to support a simplified application level view (rather than requiring the full transport level view). This is used to represent complex semantic associations between PCs where _pcPort direct association is not sufficient.

3.3.3 ConstraintDomain (CD)

Qualified Name: CoreModel::ProcessingConstructModel::ObjectClasses::ConstraintDomain

ConstraintDomain (CD) models the topological component that represents the opportunity to enable processing of information between two or more of its CdPorts.
A CdPort may be associated with another CdPort or with an LTP at a particular specific layerProtocol.
It provides the context for and constrains the formation, adjustment and removal of PCs and hence offers the potential to enable processing.
The LTPs available are those defined at the boundary of the CD.
A CD may be:
- Asymmetric such that it offers several functions and such that different functions are offered to different attached entities (e.g.,specific ViewMappingFunction).
- Symmetric such that it offers (or is considered as offering) only one function and the same function is offered to any attached entity with no interactions between attached entities.

An asymmetric CD offering a number of distinct functions will have CdPorts through which the distinct functions are accessed.
A symmetric CD offering only a single function need not have CdPorts, the function can be accessed directly from the CD.

Inherits properties from:

- GlobalClass

This class is Experimental.

Table 3: Attributes for ConstraintDomain

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_cdPort	Experimental	An asymmetric CD instance is related to LTPs via CdPorts (essentially the ports of the CD). Symmetric CDs don't have CdPorts and are directly related to LTPs.
_pcInDomain	Experimental	A CD constrains one or more PCs. A constrained PC connects LTPs that are on the CD boundary.
_ltp	Experimental	A symmetric CD instance is associated with zero or more LTP objects. The LTPs on the CD boundary provide capacity for processing. For an asymmetric CD instance the association to the LTP is via the CdPort.
_cdInDomain	Experimental	The CD class supports a recursive aggregation relationship such that the internal construction of an CD can be exposed as multiple lower level CDs. Note that the model actually represents an aggregation of lower level CDs into higher level CDs as viewpoints rather than partitions, and supports multiple views
_cascInDomain	Experimental	A controller operating in the scope defined.
_equipmentInDomain	Experimental	A ConstraintDomain can be used to represent physical constraints in the logical view. In this case the CD can be associated to the physical equipment.
_fcInDomain	Experimental	A CD constrains one or more FCs. A constrained FC abides by rules stated in the constraining CD where those rules may relate to LTPs referenced by the FC that are also included in the CD.
_fdInDomain	Experimental	A CD constrains one or more FDs. A constrained FD abides by rules stated in the constraining CD where those rules may relate to LTPs referenced by the FD that are also included in the CD.
_controlConstructInDomain	Experimental	A CD constrains one or more ControlConstructs.
_ltpInDomain	Experimental	A CD constrains one or more LTPs.
_linkInDomain	Experimental	A CD constrains one or more Links. A constrained Link connects LTPs that are on the CD boundary.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_runningOsInDomain	Experimental	A RunningOs constrained by the ConstraintDomain.
_runningSoftwareApplicationInDomain	Experimental	A RunningSoftwareApplication constrained by the ConstraintDomain.
_runningNativeVmmInDomain	Experimental	A RunningVmm constrained by the ConstraintDomain.
_fileSystemInDomain	Experimental	A FileSystem constrained by the ConstraintDomain.
_vmfInDomain	Experimental	A ViewMappingFunction constrained by the ConstraintDomain.
_partyRole	Experimental	See referenced class
_partyRoleInDomain	Experimental	See referenced class
_supportedStreamType		See referenced class
_availableStream		See referenced class
_streamProvider		See referenced class
_changeUpdater		See referenced class

3.3.4 CdPort

Qualified Name: CoreModel::ProcessingConstructModel::ObjectClasses::CdPort

The association of the CD to LTPs is direct for symmetric CDs and via CdPort for asymmetric CDs.
The CdPort class models role based access to a CD.
The capability to set up PCs between the associated CdPorts of a CD depends upon the type of CD.
It is asymmetry in this capability that brings the need for CdPort.
The CD can be considered as a component and the CdPort as a Port on that component.

Inherits properties from:

- LocalClass

This class is Experimental.

Table 4: Attributes for CdPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_cdPort	Experimental	Constraint Domains can be meshed together view their ports directly as well as via LTPs indirectly.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_ltp	Experimental	A CdPort is associated with zero or more LTP objects. The LTPs on the CD boundary provide capacity for processing. For symmetric CDs the association is directly from the CD to the LTP.
_pcPort	Experimental	Where a CD is asymmetric and hence has CdPorts and where that CD supports PCs, appropriate CdPorts of that CD support the corresponding PcPorts.

3.4 Related Classes

In version 1.3.1 the FD, FC and Equipment were listed as classes impacted by the addition of PC/CD. In version 1.3.1 FD, FC and Equipment all referred to PC and/or CD. In version 1.4 these references have been removed. This reduces model coupling with no loss of capability. References, where relevant, are now from PC/CD to other classes in the model.

4 Explanatory Figures

This section provides figures that illustrate the application of the model to various network scenarios. Further examples are provided in TR-512.A.9.

In this section, we will use some simple examples to help in understanding the intention of the ProcessingConstruct and ConstraintDomain classes, but not go into the level of detail that we will in the examples document.

4.1 Multi- Functional Device Example

As stated in the introduction, moving from a traditional Network Element oriented model to ProcessingConstruct oriented model requires us to think differently about a 'device'. Rather than thinking about a large functional block that can be 'decomposed', we should now be thinking of smaller functions that can be 'assembled'.

In general, a 'device' instance is represented by one or more ControlComponent (representing a relevant part of the control functionality of the device that are accessible through a ControlPort – see TR-512.8), one or more ControlSystemViews (where each represents a grouping of information about the device (in the form of object instance) available as a view – see TR-512.8) and one or more ConstraintDomain (where each ConstraintDomain represents a set of interrelated elements of the device) along with instances of PC, LTP, FD, FC, Equipment etc to represent various functional and physical arrangements.

In the diagram below, we show a 'device' where we have deliberately removed most of the details. The NetworkElement scope (which we assume was based on a 'chassis' physical scope in this case) has been replaced by a ControlComponent , a ControlSystemView and a ConstraintDomain. The ControlComponent and ControlSystemView are described in TR-512.8).

This particular 'device' is actively providing four functions (and a real case could have many more). Each of these functions is peering with similar functions in other 'devices' forming protocol topologies.

For the BGP control plane function we show that we can relate BGP peers by directly binding the PC ports (using PcPortBoundToPcPort) to give a simplified application view and we can also represent the transport viewpoint (via LTP, FD, FC, Link …).

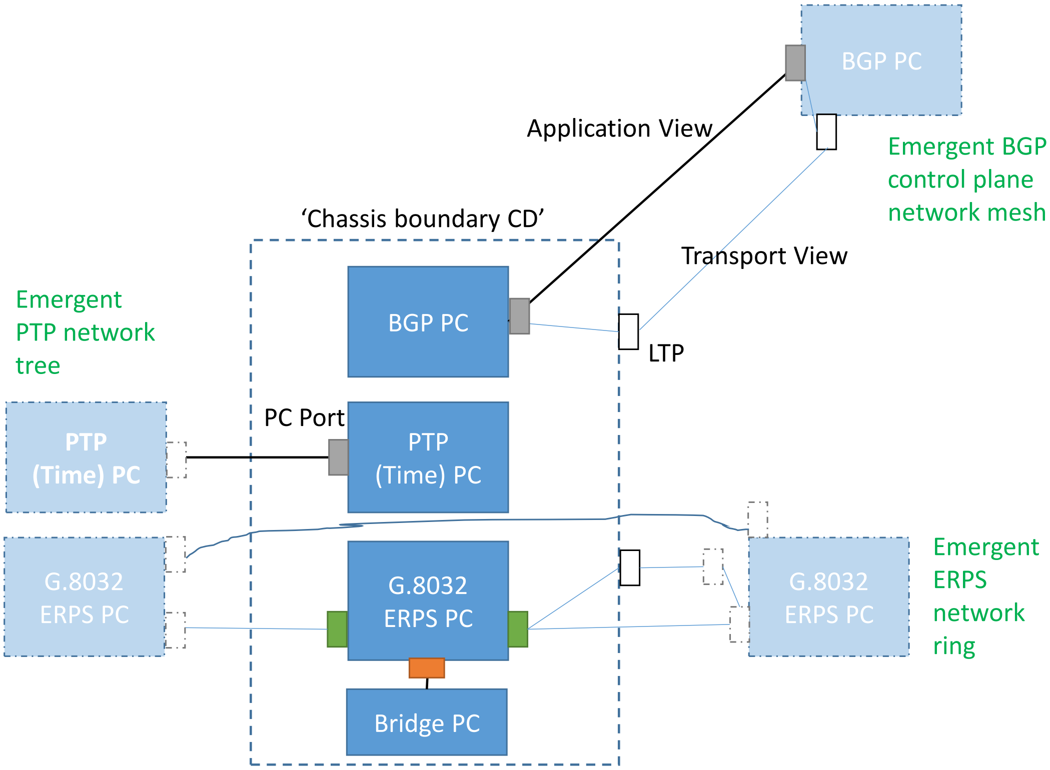


Figure 4-1 Using PC to represent complex functions and CD to constrain

4.2 Distributed Device

Because this is the overview document, we won't go through this example in detail, but it shows the generality of the ConstraintDomain concept.

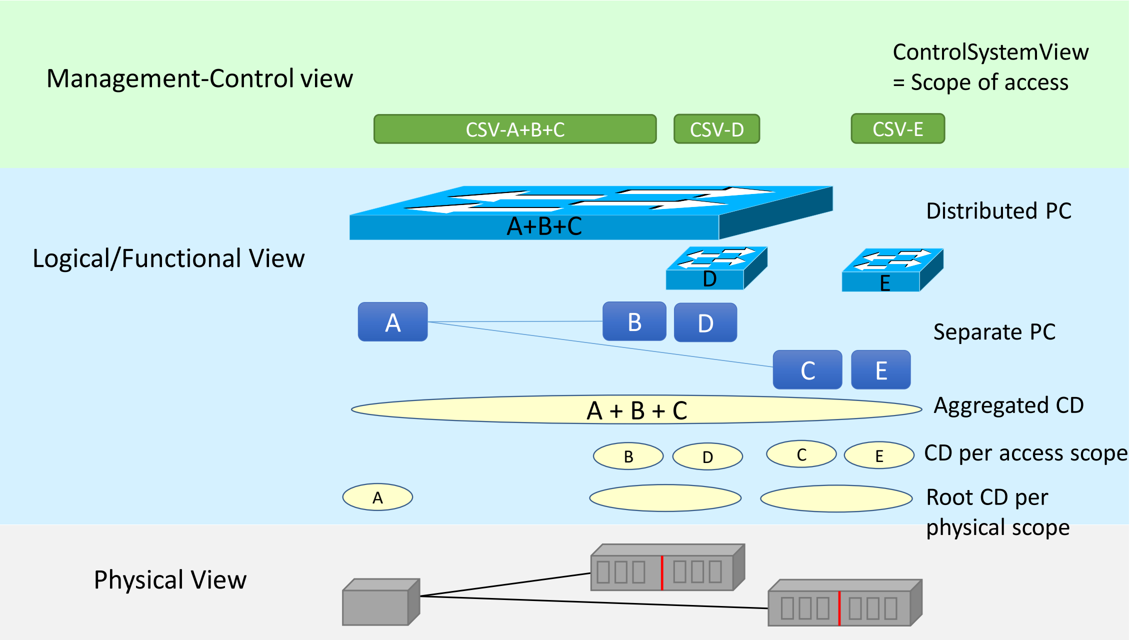


Figure 4-2 Distributed Device – Split Chassis

In the example, there are 3 chassis, with a central chassis and two satellite chassis connected. The central chassis controls only part of each satellite chassis.

At the bottom of the Logical View there is a ConstraintDomain related to each chassis, which enables enforcement of physical inventory related constraints.

For each satellite, there is a ConstraintDomain to represent the logical split in the chassis.

So, the distributed device can be viewed as a whole, there is also a Constraint Domain shown as 'Aggregated CD'.

There may also be some ProcessingConstructs that cross the chassis boundary (such as Ethernet switching) – it's just a matter of determining which constraint scope is appropriate.

Similarly, in the management plane, there may be different ControlSystemViews with different management-control scope / visibility.

4.3 PTP Clock Example

For this example, we will assume that:

- A device may have zero, one or more PTP clocks
- For each clock:
    - Clock input may be received from 1 or more foreign masters on (incoming) slave ports
    - An algorithm is used to select one of the clock inputs to use
    - This selected clock input can be propagated to zero, one or more clocks via (outgoing) master ports
    - Only ports enabled for PTP can have PTP packets extracted or injected.

Note that a port may see more than one foreign master (e.g. if there is a bridge in-between).

Note that messages are sent in the reverse direction too, this diagram just shows the clock Master – Slave relationships

This enables us to produce a simple functional view of a PTP Clock ProcessingConstruct in a 'device', as shown in the diagram below.

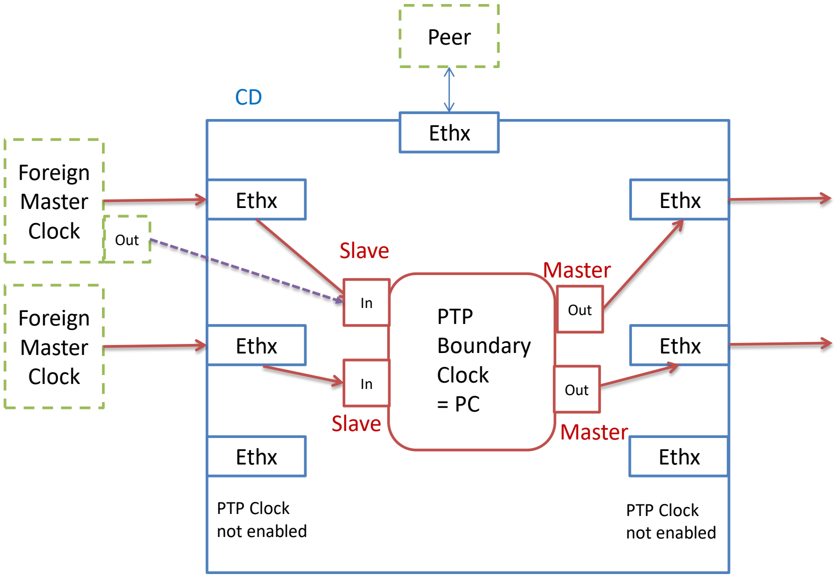


Figure 4-3 PTP Clock Concepts

We can see that the 'device' boundary is represented by a ConstraintDomain with related LTP.

Because of the PTP clock asymmetry, we need to create PcPorts and assign roles to them.

The Ethx LTP are bound to the PcPort (via association PcPortBoundToLTP).

We can also show the direct PTP topology by directly binding the PC ports (using PcPortBoundToPcPort) as shown by the purple dotted line.

Note that we are using the term 'PTP Clock' to avoid confusion of 'Precision Time Protocol' with 'Physical Termination Point'.

4.4 ERPS G.8032 Example.

Assume an Ethernet switch that can support ERPS (Ethernet Ring Protection Switching) functionality (in addition to its other functions). As before, there is a ConstraintDomain that relates to the physical boundary and related LTP.

Assume the 'device' supports a single ERPS node only, hence there is a single ConstraintDomain inside the 'device' representing the ERPS Node. An ERPS node can participate in many rings and each ring can have many instances. A ConstraintDomain represents each ring and a ProcessingConstruct represents each ring instance, nested as shown below.

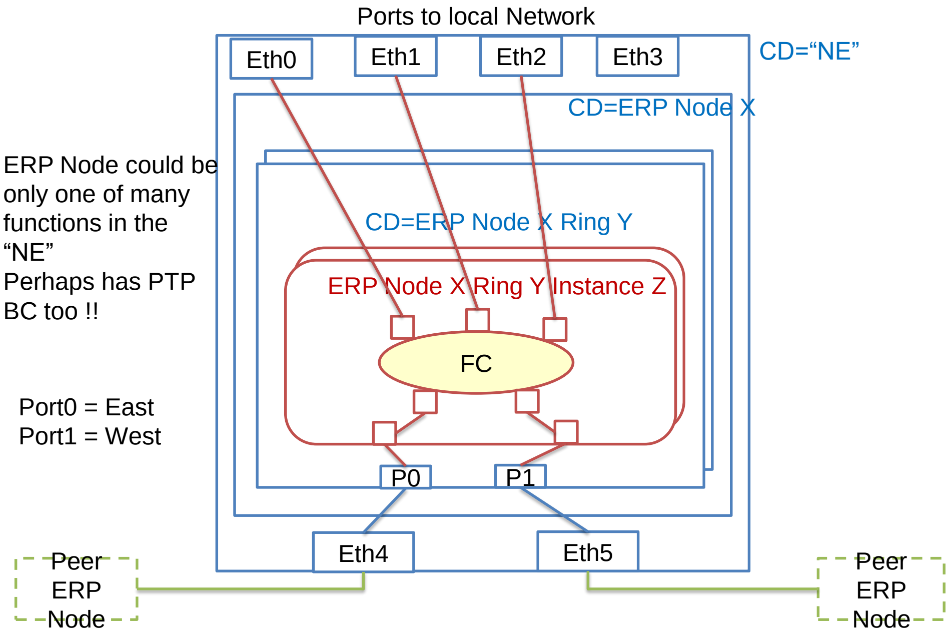


Figure 4-4 ERP [ITU-T G.8032] Concept Example

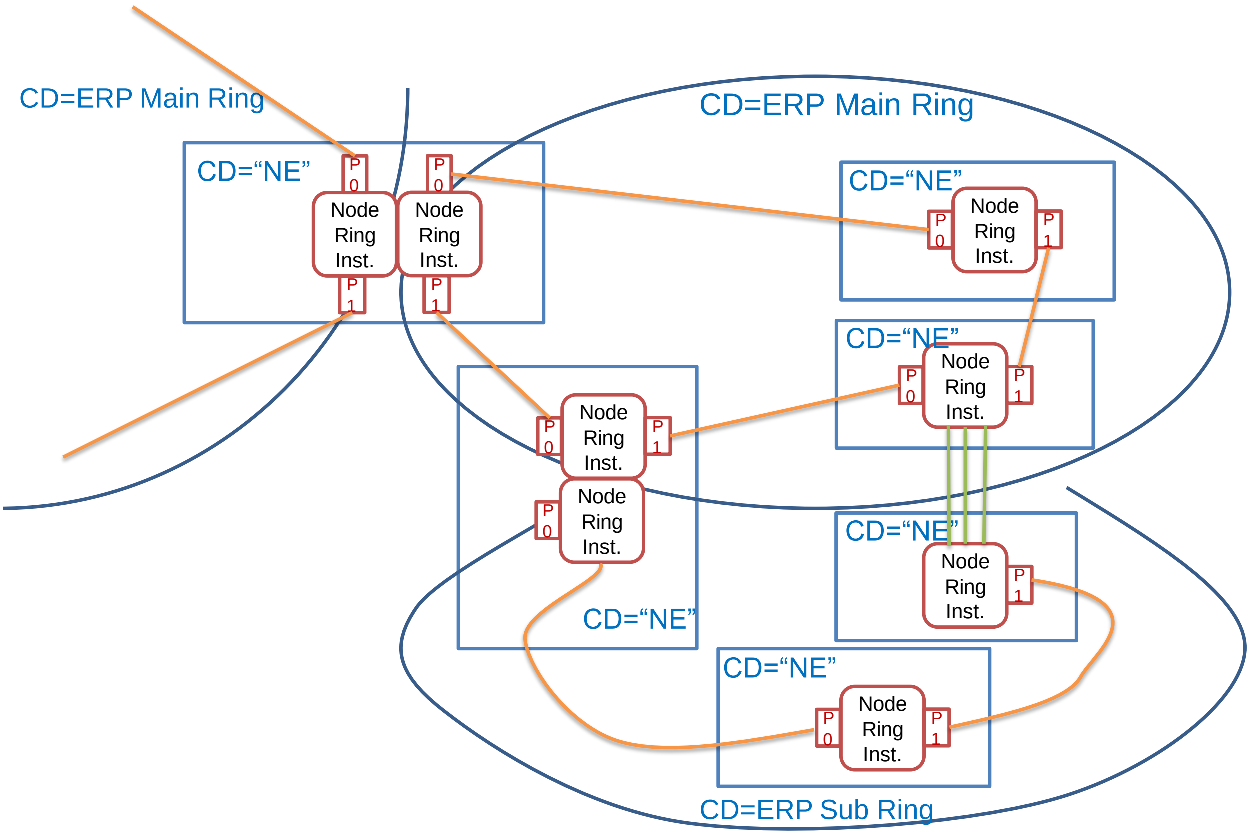
Note how the ConstraintDomain ring ports Port0 and Port1 are bound to the 'device' LTP.

Also note that inside of each ring instance in the node, there is a ForwardingConstruct that shows the ring flows.

ConstraintDomains can be used at the network level, as shown in the diagram below, to highlight ring scope.

Note also that the 'device' level constraint domains cross the ring ConstraintDomain boundaries, so this is an example where constraint domains don't form a hierarchy. This supports a 'ring viewpoint' as well as a 'device viewpoint' and shows that certain constraints are ring or 'device' scoped.

In the TR-512.5 the ConfigurationAndSwitchController is shown in place of the generalized PC as at this stage a specialist Resilience controller is use for all resilience cases.



Note NE TPE/LTP not shown. ERP node local ports not shown, R-APS channel not shown

≡ represents local port connections between ERP nodes. An ERP Node is not a "NE"

Figure 4-5 ERP Network Example

5 Further considerations (see also TR-512.FE)

1) Degree of specialization:
    - PC v ControlComponent v C&SC, PC v FC – should all functional components be PC rather than specialized?
    - CD v FD – should all domains be CD rather than specialized?
2) Further detailing the PC model:
    - Dealing with the component – system recursion
    - Projecting views
3) Modeling software to provide more precise representation of emergent PC (currently only equipment supports PC).
4) CD/PC spec and relationship to the generalized spec approach

End of document