Core Information Model (CoreModel)

TR-512.8

Control

Version 1.4
November 2018

ONF Document Type: Technical Recommendation
ONF Document Name: Core Information Model version 1.4

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	Major rework of model.

1 Introduction

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 the Control model

As explained in TR-512 V1.2 the classes SdnController, NetworkControlDomain and NetworkElement[1] have been reassessed and deprecated and new classes have been developed in

[1] The Network Element scope of the direct interface from a SDN controller to a Network Element in the infrastructure layer is similar to the EMS-to-NE management interface defined in the information models [ITU-T G.874.1] (OTN), [ITU-T G.8052] (Ethernet), and draft [ITU-T G.8152] (MPLS-TP).

this release to replace them. It has been recognized that a uniform recursive model of control can be developed that provides a consistent treatment of what were previously seen as completely different things.

This document describes a general model of control suitable for representation of the capabilities that control the network and for representation of the relationship to the model of the network from the control perspective. The document also discusses the dismantling of the NE and recasting aspects of it as Control.

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

3 Model of control component and views

3.1 Background

The ONF Architecture [ONF TR-521] talks of a recursion of control aligning well with the more general concept of the Management-Control Continuum from [TMF IG1118]. Similarly, [ITU-T G.7702] also describes recursive arrangements of SDN controllers. The control model in [ONF TR-512 V1.2] showed a traditional hierarchy rather than a generalized recursion.

Over many years it has become apparent that the traditional representation of Network Element (NE) and of Managed Element (ME) was not correct (see section 4.7 Dismantling the NE – Some rationale on page 36 for more detail and TR-512.A.7 for examples). It is clear that from one perspective the Network Element is simply a lower level member of the Management-Control Continuum. It is also apparent that all other aspects of the NE are covered by other parts of the model.

It was concluded that the NE should be remodeled. The remodeling was driven a rational separation of concerns. During the work, the network element concept logical functions (PC, FC, LTP etc.) and physical structure (Equipment etc.) were split off. What was left was the network element control function.

The two things needed to represent the control function are:

- The (logical) location of control functions in the network and how they are related (control network)
- The scope of network functions that each control function controls

The decision was made to create a separate control function class ControlConstruct and reuse the ConstraintDomain class for the control scope representation. Reusing ConstraintDomain simplified the resulting model (otherwise a lot of associations would have needed to be duplicated).

It then became apparent that this general model could also be used to model other functional groupings e.g. an SDN controller, giving a consistent view of the different elements in the control network and that that capability should be generalized so that it could handle all aspects of the Management Control Continuum.

The model chosen for the Control functions is derived from the Component-System pattern (see TR-512.A.2) and the ProcessingConstruct (see TR-512.11 and TR-512.A.9). It was then clear that as a controller controls components then the components of the controller that deal with controlling other things also needed to be controlled (as is explained in the Management Control Continuum). That is, MCC functions themselves can be managed/controlled.

The following sections set out the model in this context.

3.2 The control model in the context of the core classes

It is useful to categorize the functions in a network in terms of the type of functions that they provide. Two key function types are processing and transport of information.

The figure below shows the key model entities and the functions that they perform.

Key Class	Processing Function	Transport Function	Constraint	Reference
LogicalTerminationPoint (LTP)	✓ - protocol stack termination (Transform)	-	✓ - client creation	TR-512.2
ForwardingConstruct (FC)	-	✓ - forwarding (Transfer)	✓ - bounded forwarding	TR-512.2
ForwardingDomain (FD)	-	-	✓ - FC creation, LTP creation	TR-512.4
Link	-	-	✓ - FC creation, LTP creation	TR-512.4
ControlConstruct (CC)	✓ - management-control plane (communications)	-	-	TR-512.8
ConfigurationAndSwitch Control (CASC)	✓ - management-control plane (control)	-	-	TR-512.5
ConstraintDomain (CD)	-	-	✓ - general constraints (augmenting above)	TR-512.11
ProcessingConstruct (PC)	✓ - any hybrid functions and any other function not above	-	-	TR-512.11

- = insignificant (may be non-zero – e.g. all Processing Functions are bound to encapsulate some forwarding and it can be argued that forwarding is a form of processing)
There is a 3rd function , Storage that isn't supported by any of these

Figure 3-1 Key entities in the model

Both ProcessingConstruct and ControlConstruct perform processing functions, but while a ProcessingConstruct just processes its information, a ControlConstruct processes information to control other functions (such as ProcessingConstructs, Forwarding constructs etc.). It is this additional controlling responsibility that means that it makes sense to have a separate model entity for ControlConstruct.

As discussed (in TR-512.11, TR-512.2 etc.), the concept of NetworkElement has been removed from the model. The model now focusses on network functions and the boundaries that they operate within (ConstraintDomain). This section works through some basic examples to introduce the concepts of the model prior to embarking on the description of the model itself. Some of the figures used in this section are further discussed in TR-512.A.7).

The figure below shows a simple representation of a NetworkElement on the left. On the right, a Control Construct has been added and a ConstraintDomain to represent the scope of control.

Note that:

- The ControlConstruct itself exists within a ConstraintDomain boundary and uses another ConstraintDomain boundary to show the scope of its control.
- To keep the diagram uncluttered, "PC etc" stands in for LTP, FC, FD, Equipment, RunningSoftwareApplication, etc.

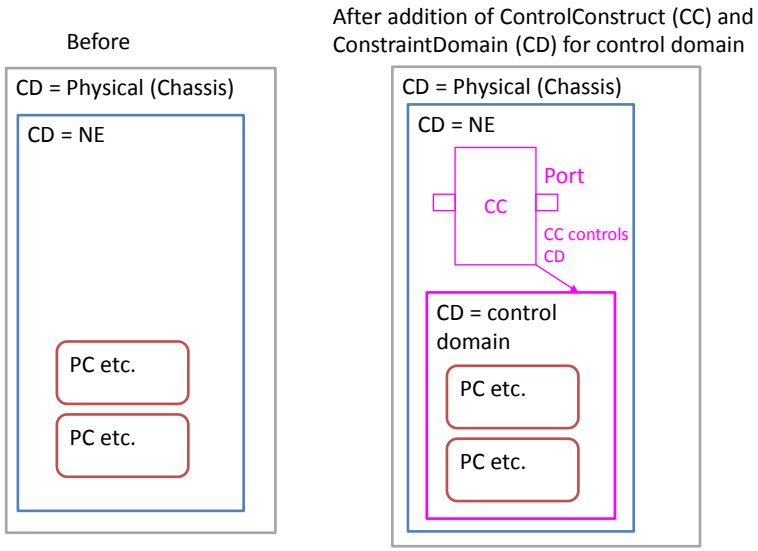
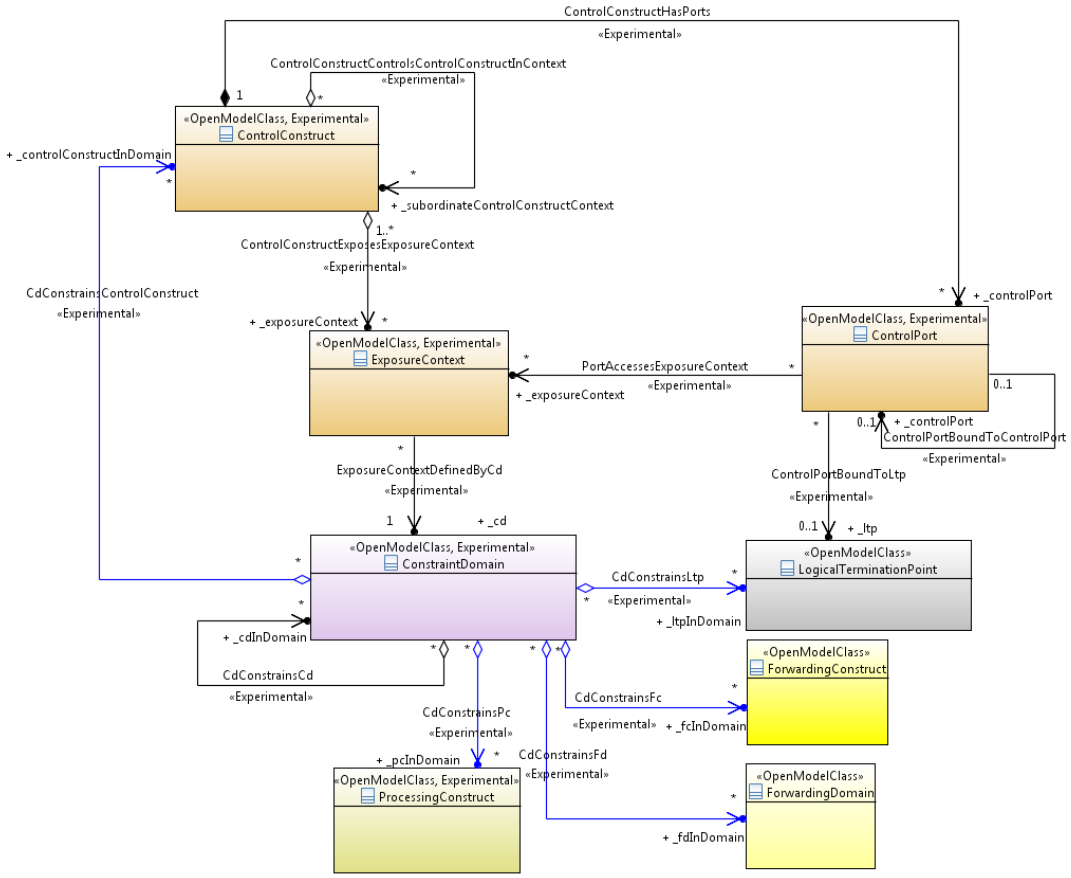


Figure 3-2 – Basic Network Element

The model also needs to be able to represent a control network, and this is achieved in two ways:

1. By representing the binding between ControlConstruct ports
2. By the nesting of ControlConstructs in a ConstraintDomain that is controlled by another ControlConstruct

The figure below shows as summary of the Control model.



CoreModel diagram: Control-ControlConstructSummary

Figure 3-3 Core Control Model Summary

The basic layering concept is shown in the diagram below.

Also note that because of the ControlPortBoundToLtp association, the logical control port bindings can also be linked to any transport representation if appropriate.

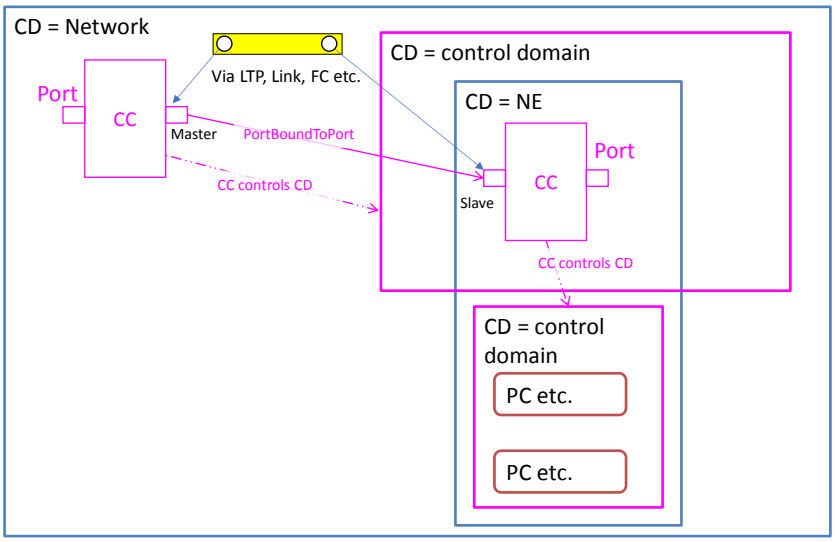


Figure 3-4 - Basic ControlConstruct layering Use Case

Note that while the architecture and model allow for the control network to extend down to show the control port bindings to all the network functions (as discussed for the Component-Port pattern in TR-512.A.2), the bindings can be implied from the control domain. Within a device there is no associated transport requirements and hence these bindings can be omitted in an implementation, reducing the complexity of the information stored.

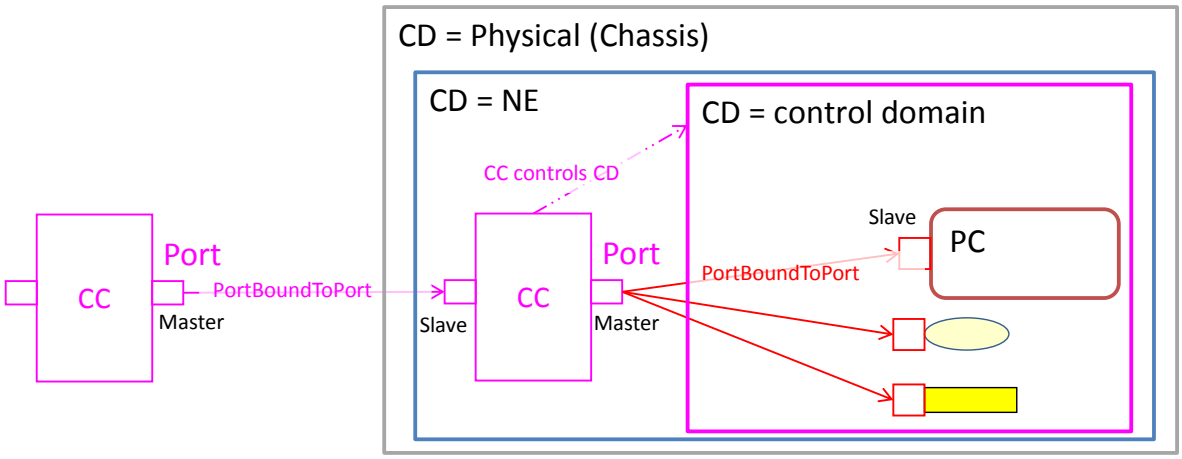


Figure 3-5 - Control port to "PC etc." port binding

It would be possible to add ports to every construct for control purposes and then bind these to the CC ports. This makes sense architecturally and provides good consistency bu:

- Locally within an NE, the binding is usually implied rather than explicitly defined/managed/controlled (e.g., a BGP process is defined via the CC so its binding is implicit)

- It adds a lot of complexity to the instance graph, to create and manage all these ports and bindings
- Since it is expected that a local management/control agent to be present, the bindings are local, so there are no transport (via FC) implications

The model can be used to represent an SDN controller consistent with the ONF architecture [ONF TR-502] and [ONF TR-521] using the same classes used to represent a Network Element. The controller boundary is represented using a ConstraintDomain and the functions inside are represented using ProcessingConstructs etc. This is discussed in detail in TR-512.A.7.

The model can represent the controller groupings and layering. The two diagrams below show some possible ways that this could be achieved. Note that the model doesn't enforce any particular controller architecture, it just supports the general concepts.

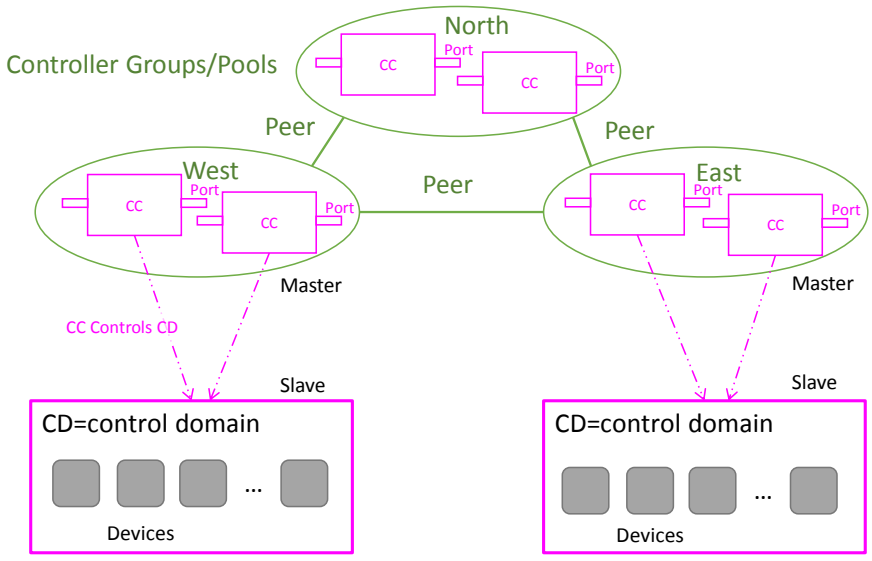


Figure 3-6 - A mix of Master-Slave and Peering

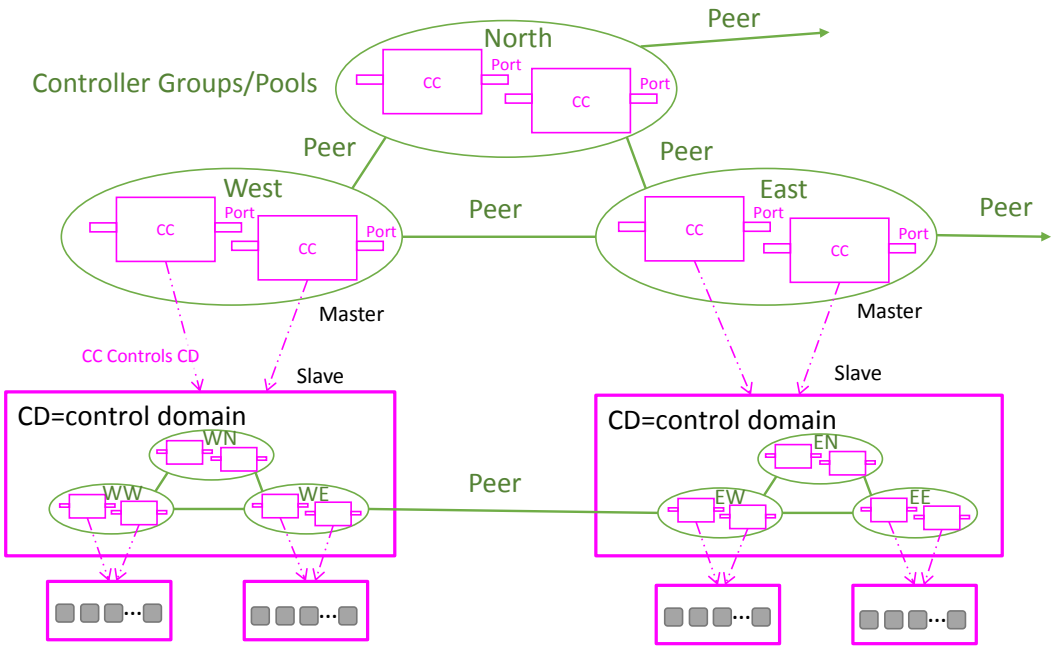


Figure 3-7 - Recursive Control Architecture

An ExposureContext instance defines what can be accessed through a particular ControlPort and who can have access (for more information refer to later sections in this document). The ExposureContext allows a ControlConstruct to give another ControlConstruct access to some view of the network functions that it is controlling. For example, as shown in the figure below, ControlConstruct X wants to give ControlConstruct Y access to ProcessingConstructs 3 and 4. The relevant view is defined by a ConstraintDomain.

In the example, if there hadn't been an existing ConstraintDomain with just PC-3 and PC-4, then a new ConstraintDomain would have been created and the ProcessingConstructs added to it. The ExposureContext then links the exposing ControlConstruct, the exposed scope and the receiving ControlConstruct together.

Note that ExposureSession can be considered to be a form of security access, so it may:

- Require authentication of the ControlConstruct that the functions are exposed to
- Be for a limited time span
- Limit the authorized actions that can be performed on the exposed network functions (read, modify, delete) by the ControlConstruct they are exposed to

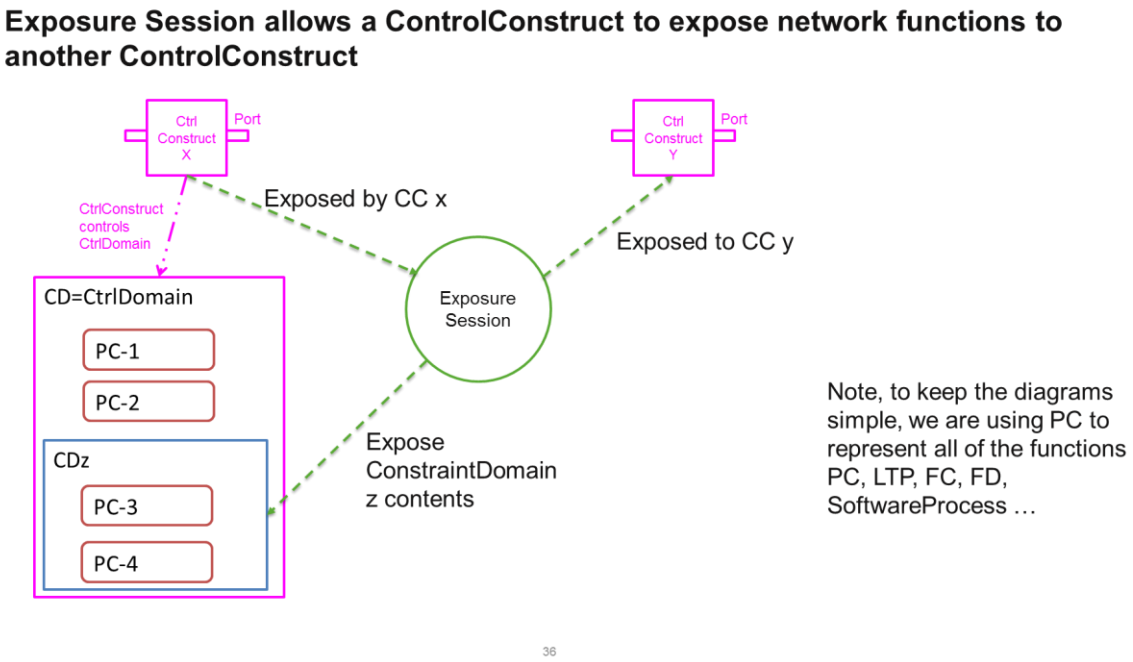
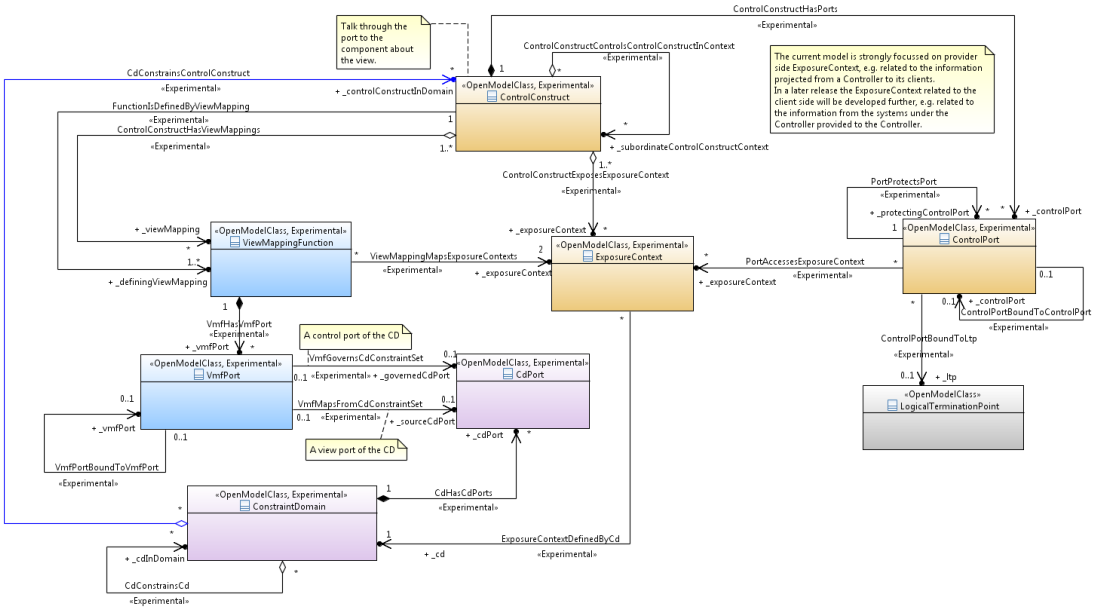


Figure 3-8

Note that further work needs to be done on the remaining part of the model to provide network function and name mappings and this could replace the ViewMappingFunction and its port in a future release.

3.3 The control model in the context of the core classes

The figure below shows the core of the Control model.



CoreModel diagram: Control-ControlConstructAndExposureContextCore

Figure 3-9 Core Control Model

The classes are described in the section below. Some aspects of the model described below are shown in figures in sections 3.5, 3.6 and 3.7. The figures above intentionally do not include all associations etc. mentioned in the detailed class information below. The figures focus on the control model, the classes listed show all aspects of the class.

3.3.1 ControlConstruct

Qualified Name: CoreModel::GeneralControllerModel::ObjectClasses::ControlConstruct

Represents control capability/functionality.
ControlConstructs communicate with other ControlConstructs through ControlPorts about things within the related ConstraintDomains.
The ControlConstruct applies to all Control/Management cases including:
- the controller in the Network/Managed Element e.g. an SNMP agent).
- an SDN Controller.
- an EMS.
- an NMS.
This specific model follows a subset of the Component-System Pattern.

This class is Experimental.

Table 1: Attributes for ControlConstruct

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_exposureContext	Experimental	A view supported by the ControlConstruct that may be exposed at a ControlPort of the ControlConstruct.
_definingViewMapping	Experimental	ControlConstruct behavior is defined in part by view mappings.
_controlPort	Experimental	A port on the ControlConstruct that allows access to the functions of the ControlConstruct.
_subordinateControlConstructContext	Experimental	A ControlConstruct that is part of an abstract view of the system that supports the referencing ControlConstruct and hence describes part of the behavior of the referencing ControlConstruct.
_viewMapping	Experimental	ControlConstruct uses the referenced ViewMapping to produce one view from another.
_controlTasks	Experimental	An activity being carried out by the ControlConstruct where that activity is being exposed such that progress can be observed through a ControlPort.

3.3.2 ControlPort

Qualified Name: CoreModel::GeneralControllerModel::ObjectClasses::ControlPort

The access to the ControlConstruct following the normal Component-Port pattern (i.e., the functions of a component are accessed via ports).
Is assumed to usually be bidirectional.
This class is Experimental.

Table 2: Attributes for ControlPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_protectingControlPort	Experimental	A simple representation of resilience where one ControlPorts are identified as providing equivalent information.
_controlPort	Experimental	Control Ports may be used to associate controllers in a hierarchy and as peers. Peer controllers are assumed to both the subordinate of each other.
_ltp	Experimental	The LTP through which the control messaging/signaling flows.
_providerRole	Experimental	Properties relevant when the ControlPort is exposing the ControlConstruct as a provider of capability.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_userRole	Experimental	Properties relevant when the ControlPort is exposing the ControlConstruct as a user of capability.
_exposureContext	Experimental	A view presented through the ControlPort.

3.3.3 ExposureContext[2]

Qualified Name: CoreModel::GeneralControllerModel::ObjectClasses::ExposureContext

A view of the things controlled by a control system. For example, a virtual network of ONF TR-502, or more generally, resources (clause A.10 of ONF TR-521).
A referenced ConstraintDomain bounds a view which is a structured presentation of the underlying controlled things (the "actual" entities) for some purpose.
The model bounded by the ConstraintDomain is constructed by mapping/abstracting the models of the underlying controlled things.
The ControlConstruct is itself controlled and presents itself in terms of ControlConstructs (subordinate) in a view.
At one extreme the referenced ConstraintDomain may expose all underlying details of everything controlled with no adjustment from the presentation provided by the controlled things.
A ConstraintDomain may expose a subset of the controlled things that focuses on a particular aspect (e.g., only the ControlConstructs).
A ControlPort has an association to the ExposureContext that explains, via the related ConstraintDomain, what can be acquired through the port
The emphasis is on exposing a constrained set of information and operations
Bounds what is presented over an interface from a particular viewpoint. The domain of control is almost always broader than the entities etc. bounded by the ConstraintDomain.
Represents the domain of control available to the viewer.
This class is Experimental.

Table 3: Attributes for ExposureContext

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_cd	Experimental	The ConstraintDomain that defines the view to be exposed.

[2] The explicit class, ControlSystemView, that was used in 1.3.1 has been replaces with ExposureContext and associated general ConstraintDomain class. There may be further refinements in this area.

3.3.4 ConstraintDomain

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.
- 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 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 4: 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 asymmetric FDs the association to the LTP is via the FdPort.
_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.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_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 connects LTPs that are on the CD boundary.
_fdInDomain	Experimental	A CD constrains one or more FDs. A constrained FD connects LTPs that are on the CD boundary.
_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.
_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.

3.3.5 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 5: 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.
_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.

3.3.6 ViewMappingFunction

Qualified Name: CoreModel::GeneralControllerModel::ObjectClasses::ViewMappingFunction

The rules that relate one view to another.
A ControlConstruct aggregates ViewMappingFunctions.
Each ViewMappingFunction in the context of a ControlConstruct define the relationship between the views presented in an ExposureContext of that ControlConstruct and other views within the Controller.
The ViewMappingFunction is applied to the entities aggregated by one or more ConstraintDomains (via VmfPort - CdPort VmfMapsFromCdConstraintSet to construct the view in another ConstraintDomain (via VmfPort - CdPort VmfGovernsCdConstraintSet association).
For example, a pair of LTPs with matching adjacency tags in a nodal view may be mapped to a Link in a network view where the rules would describe the matching criteria etc.
This class is Experimental.

Table 6: Attributes for ViewMappingFunction

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_exposureContext	Experimental	An ExposureContext available to the ViewMappingFunction.
_vmfPort	Experimental	A port of the ViewMappingFunction.

3.3.7 VmfPort

Qualified Name: CoreModel::GeneralControllerModel::ObjectClasses::VmfPort

A port of the MappingFunction through which the effects of the mapping is exposed.
This can be an input to the mapping of as an output of the mapping where the inputs and outputs may have more detailed roles.
This class is Experimental.

Table 7: Attributes for VmfPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_vmfPort	Experimental	Feeding to/from another Vmf.
_sourceCdPort	Experimental	Drawing from a ConstraintDomain that aggregates classes to feed the mapping.
_governedCdPort	Experimental	Causing instances of classes to be created/deleted/modified in the context of a ConstraintDomain that aggregates a view. This governs what the ConstraintDomain may aggregate and also governs the lifecycle of the aggregated entities.

3.4 Further description

A ControlConstruct instance may expose, through each associated ControlPort instance, one or more views of controlled instances (i.e., instances of FC, LTP etc.). A view provided via a ControlPort instance is defined by an ExposureContext instance. The controlled instances to be exposed in a view are aggregated by a ConstraintDomain instance referenced by the ExposureContext instance defining the view.

A ControlConstruct instance may provide different views, each specified via a separate ExposureContext instance, via different ControlPorts instances. Several ControlPorts instances of a ControlConstruct instance may relate to the same ExposureContext instance and will hence expose the same view.

Several ExposureContext instances may reference the same ConstraintDomain instance and hence may provide the same view and several ControlConstruct instances may reference the same ExposureContext instance and will therefore expose the same view through at least one of their port instances.

The structure of the instances of the classes aggregated by a ConstraintDomain (the output view) may be derived from the structure of the instances of the classes aggregated by one or more other ConstraintDomains (input views). The inter-view mapping/abstraction/refactoring rules are maintained by one or more ViewMappingFunction instances that reference the ExposureContext instance. A ViewMappingFunction determines the specific instances in the view and hence determines the instances of FC, LTP etc. to be aggregated by the ConstraintDomain.

The derivation method will be such that an instance from an input view may be split into many instances in the output view, several instances from one or more input views may be pruned and combined to form an instance in the output view etc. The view construction is governed by

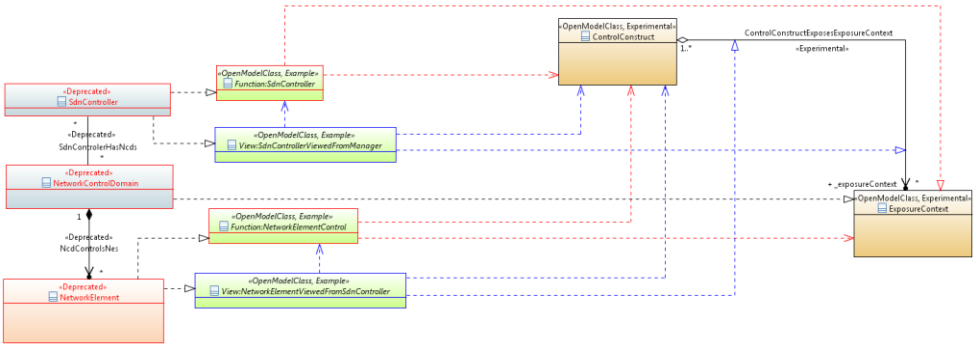
constraints and corresponding behavior (rules, policy, functions). The mapping/abstractions/refactoring may lead to new insight.

For example, a "Network Element" may have a property recorded against a port where that property was extracted from an incoming signal and where that property is defined as some form of discovery tag which, unknown to the NE has been sent by another NE. As the overarching controller can see both NEs (amongst many others) it can determine the interconnectivity from these two tags. The model of "port and discovery tag" can be refactored to an off-network link and then an off-network link pair can be refactored (combined) to be a Link where the instance combination is driven by matching discovery tags. As a consequence, new insight of interconnectivity is achieved.

A combination of ViewMappingFunctions would provide the class model refactoring rules from ExposureContext to ExposureContext and, therefore, the instance refactoring rules.

3.5 Relationship to TR-512 V1.2 model

The relationship between the V1.2 classes (that have been deprecated) and the V1.4 classes is depicted in the figure below.



CoreModel diagram: Control-MappingToControlConstructAndExposureContext

Figure 3-10 Mapping Core Control Model to traditional view

The V1.2 classes are shown with (red text and a red border). These are related to the V1.4 classes (shown with black text and a black border) via some explanatory classes (shown with a green fill). The relationships are purely pictorial.

The explanatory classes show (via the black dashed associations) that:

- The SdnController class (of V1.2) represents both the SDN Controller function and a view of that function as seen through an interface provided by a manager of the SDN Controller
- The NetworkControlDomain (of V1.2) represents the view of the network controlled by the SDN Controller as presented by the SDN Controller
- The NetworkElement (of V1.2) represents the embedded Network Element Control function presented to the SDN Controller as well as a view of that function as seen through an interface provided by the SDN Controller controlling the NE

The dashed associations, red for Functions and blue for views, highlight (roughly) that in the V1.4 model:

- The NetworkElementControl function is represented by a ControlConstruct and corresponding ExposureContext and ConstraintDomain which will have:
    - LTPs, FCs and other abstract representations of NE functions
    - Any relevant ControlConstructs that make up the control functions of the NE, such as log managers and alarm queue functions, of the NE[3]
- The SdnController function is represented by a ControlConstruct and corresponding ExposureContext and ConstraintDomain. The ConstraintDomain will have:
    - ControlConstructs representing the Network Elements controlled by the SDN Controller (see NetworkElementViewedFromSdnController below)
    - LTPs, FCs and other abstract representations of network functions abstracted from the assembly of NE level functions
    - Any relevant ControlConstructs that make up the control functions of the SDN Controller, such as log managers etc.
- The NetworkElementViewedFromSdnController view will include:
    - A ControlConstruct, ExposureContext and ConstraintDomain representing the NE as relevant to the specific view provided by the SDN Controller
        - The ConstraintDomain will have:
            - LTPs, FCs and other representations of NE functions
            - Any relevant ControlConstructs that make up the control functions of the NE, such as log managers and alarm queue functions, of the NE to be exposed

        Where the instances in the view are all abstractions (pruned and refactored forms) of those provided by the actual NE
- The SdnControllerViewedFromManager view will include:
    - A ControlConstruct, ExposureContext and ConstraintDomain representing the SDN Controller as relevant to the specific view provided by the Manager (seen through an interface provided by the manager managing the SDN Controller)
        - The ConstraintDomain which will have:
            - LTPs, FCs and other abstract representations of network functions (see SdnController above)
            - Any relevant ControlConstructs that make up the control functions of the SDN Controller (see SdnController) above
            - ControlConstructs representing the Network Elements controlled by the SDN Controller (see NetworkElementViewedFromSdnController below)

    Where the instances in an ExposureContext are all abstractions (pruned and refactored forms) of those provided by the actual SDN Controller

Clearly the above is recursive and hence a Manager could present the following via the same mechanism:

[3] The model does not provide explicit representations for such ControlConstructs. Instances of the generalized ControlConstruct class (or of the Casc class) should be used decorated appropriately.

- A ControlConstruct representing the manager itself
- A ControlConstruct representing each subordinate manager
- A ControlConstruct representing each SDN Controller subordinate to each subordinate manager
- A ControlConstruct representing each NE controlled by each SDN Controller…

A complex NE could represent subordinate parts again through the same mechanism leading to a deep Component-View hierarchy.

The classes listed here are provided in the model to assist in the understanding of the mapping from ManagedElement, SdnController and NetworkControlDomain.

3.5.1 Function:NetworkElementControl

Qualified Name:
CoreModel::GeneralControllerModel::ObjectClasses::ExplanatoryClasses::Function:NetworkElementControl

Traditional model of the NE equivalent to an aspect of the NetworkElement class from v1.2.
This class should not be implemented.
This class is abstract.

This class is Example.

3.5.2 Function:SdnController

Qualified Name:
CoreModel::GeneralControllerModel::ObjectClasses::ExplanatoryClasses::Function:SdnController

Traditional model of the SDN controller equivalent to the SdnController class from v1.2.
This class should not be implemented.
This class is abstract.

This class is Example.

3.5.3 View:NetworkElementViewedFromSdnController

Qualified Name:
CoreModel::GeneralControllerModel::ObjectClasses::ExplanatoryClasses::View:NetworkElementViewedFromSdnController

Traditional model of the view of the NE controller as seen from a SDN Controller equivalent to an aspect of the NetworkElement class from v1.2.
This class should not be implemented.
This class is abstract.

This class is Example.

3.5.4 View:SdnControllerViewedFromManager

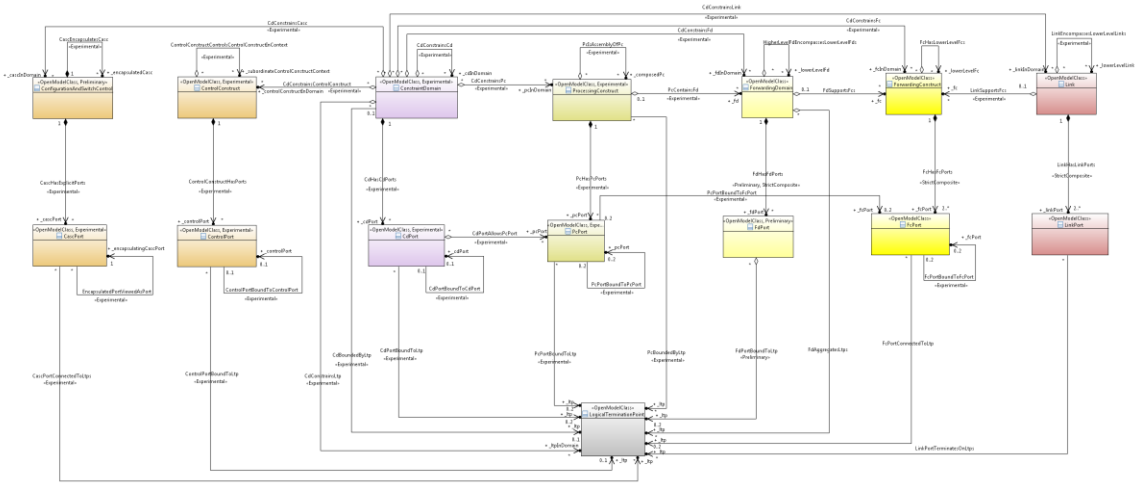
Qualified Name:
CoreModel::GeneralControllerModel::ObjectClasses::ExplanatoryClasses::View:SdnController ViewedFromManager

Traditional model of the view of the SDN controller as seen from a manager .
No equivalent in v1.2.
This class should not be implemented.
This class is abstract.

This class is Example.

3.6 Relationship to the other key classes

The following figure shows the relationship between the key Control classes and the other key classes of the model. The structural similarity is illustrated by positioning as there is no formal mechanism for enforcing patterns (e.g., inheritance does not express the pattern or enforce the constraints). The relationship is essentially the adoption of the pattern.

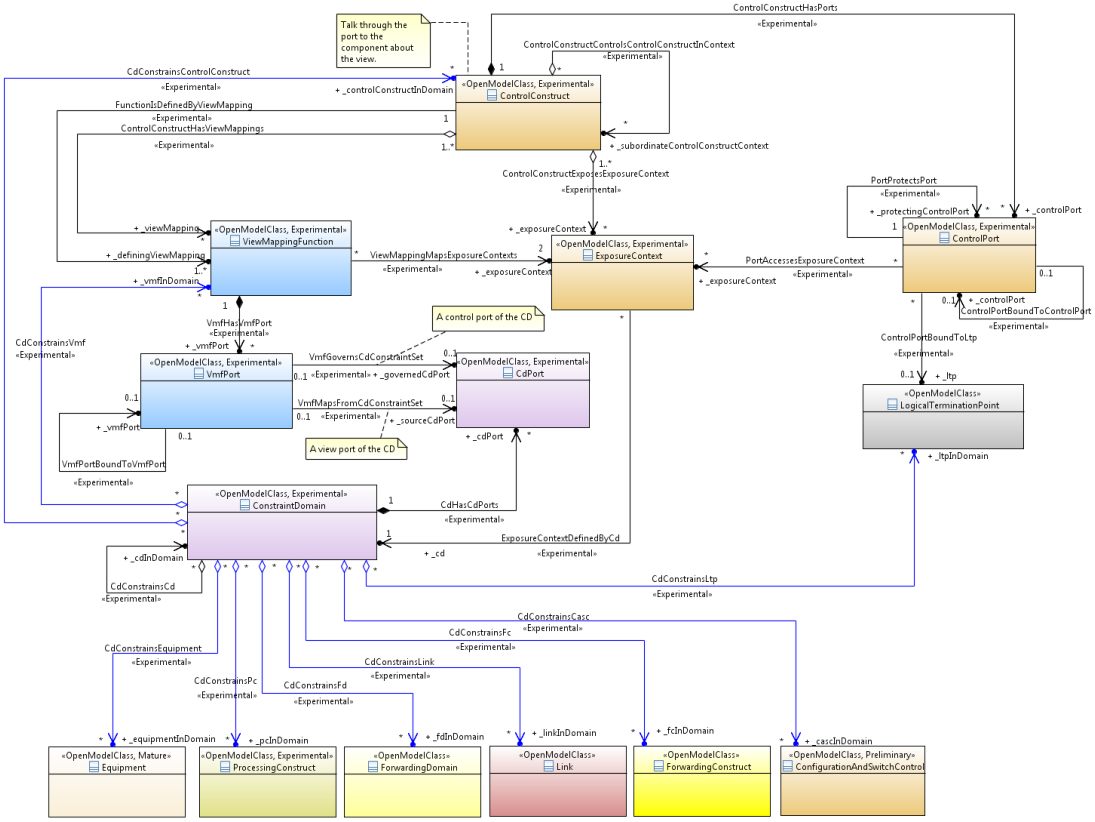


CoreModel diagram: Control-ControlConstructPattern

Figure 3-11 Relationship of Control Model to ProcessingConstruct

3.7 Model in context – directly controlled things

The figure below shows each of the key classes as potential members of one or more ConstraintDomains via the "CdConstrains…" associations (highlighted in blue).



CoreModel diagram: Control-ControlConstructFullModel

Figure 3-12 Control Model showing Controlled Entities

In the figure above several classes are shown at the bottom of the diagram aggregated in the ConstraintDomain. These are described in detail in other documents. Most notable, is the ConfigurationAndSwitchController (C&SC) which is a low-level controller, this class is described in detail in TR-512.5.

3.8 General discussion

The key consideration here is that the ControlConstruct exposes one or more ExposureContexts (the replacement for the NetworkControlDomain etc.) which include, via associated ConstraintDomain, an aggregation of all relevant controlled entities (where a controlled entity is allowed to be in many ExposureContexts).

The model is best illustrated by considering the figure below which depicts an SDN Controller controlling two devices. The white numbers in blue circles are used in the description below the figure.

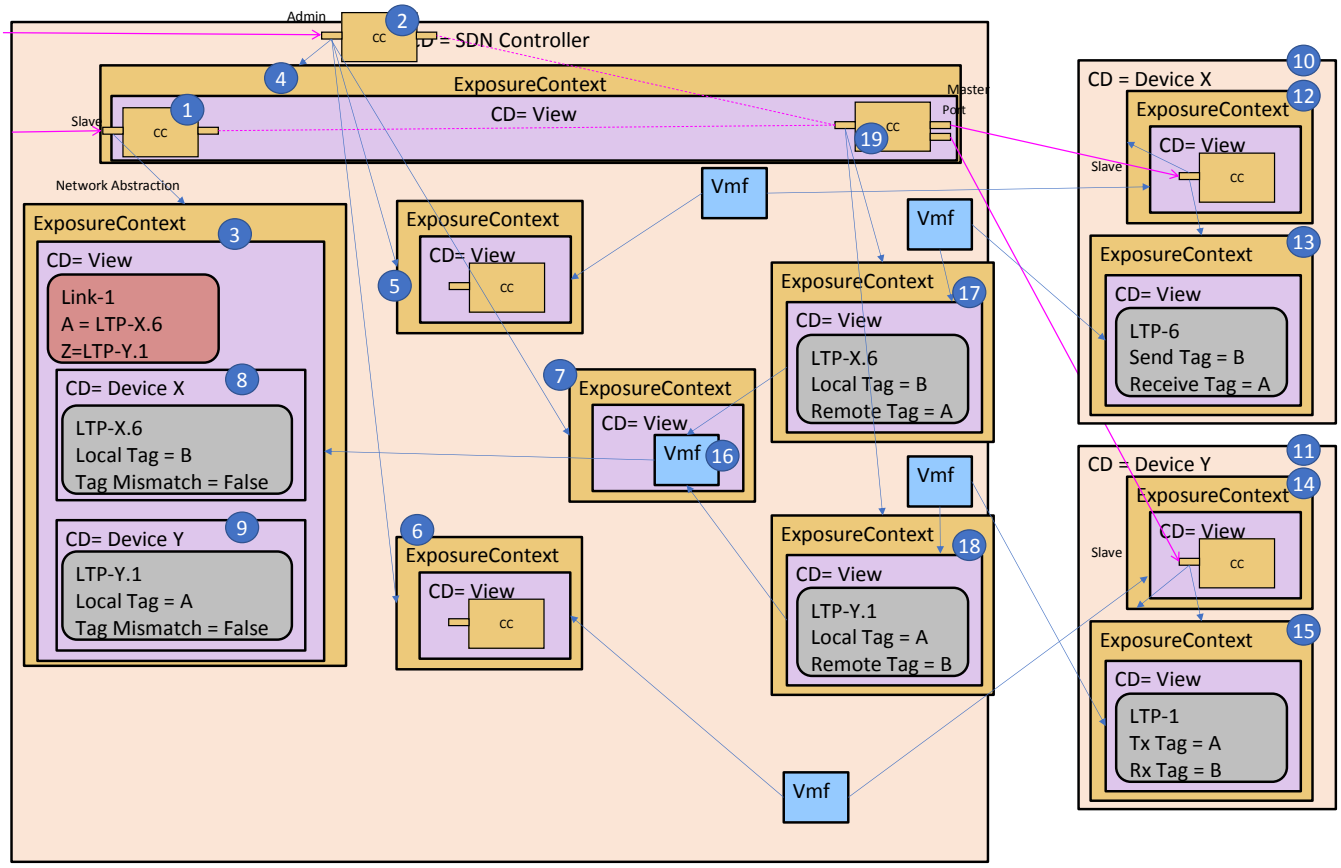


Figure 3-13 – SDN Controller controlling two devices

The SDN Controller function/scope, is represented by a ConstraintDomain, essentially the SdnController in V1.2. The SDN Controller exposes its behavior via a set of ControlConstructs (1 and 2). These provide various views, defined by ExposureContexts and corresponding ConstraintDomains (3 -7). These are exposed through ControlPorts of the ControlConstructs:

- The network view (3) of the behaviour of the devices it controls.
  - The ExposureContext has a ConstraintDomain that aggregates the purely network aspects and subordinate ConstraintDomains (8 and 9), essentially the V1.2 NetworkElement, that aggregate the relevant nodal aspects of the devices that it Controls.
  - The view will include a all FDs, FCs, Links etc. at the network level and also all LTPs, FCs etc. at the nodal level where the LTPs and FCs are associated as described in TR-512.2, TR-512.4 etc.
- The control behaviour (5 and 6) of the devices it controls.
  - The ExposureContext has a ConstraintDomain that aggregates a mapping from the ControlConstruct of the Device, i.e., the control aspects of the Network Element – the NetworkElement in V1.2.
  - The view may include properties related to alarm queues etc. on the device.
- Modifiable ViewMappingFunctions (16) of the SDN Controller.
  - The ExposureContext (7) has a ConstraintDomain that aggregates one or more ViewMappingFunctions

    - The ViewMappingFunction will expose both the fixed and adjustable aspects of the view mapping and in the case depicted would provide details of the transformation from device to network view.
- The control behaviour (4) of the SDN Controller itself.
    - The ExposureContext has a ConstraintDomain that aggregates the adjustable ControlConstructs of the SDN Controller.
    - The view may include properties related to queues etc. in the SDN Controller.

In general, the SDN controller presents mappings of some, but not all, of the capabilities of the devices (10 and 11) it controls and mappings of some, but not all, of its own capabilities via various ControlPorts. These are presented using the classes of the ONF CIM or of any other relevant model (e.g., [TAPI]) at the ports of the relevant ControlConstructs (1 and 2). The capabilities of the device exposed via relevant ExposureContexts (12 – 15) could be presented using the ONF CIM, as depicted, but most probably will use some other model such as that of TL1, [ONF OpenFlow][4] or [OpenConfig].

Consider a control function (a ControlConstruct) in a device tasked with the control of a function terminating a stream of packets (a termination function). If the device was using the ONF Core Model, the control function of the device will present an ExposureContext (13 and 15) which includes, via the associated ConstraintDomain, an LTP that in part represents the termination function. The control function will also represent its own capabilities (perhaps a capability to notify) via other view entities, not detailed here, along with a ControlConstructs, aggregated by the ConstraintDomain associated with the ExposureContext. An example of such a control function is a Network Element SNMP agent (see section 4.1 Rationale on page 30).

As discussed, a ControlConstruct representing an SDN Controller can present a network level ExposureContext (3) of the functions of the network of devices that it controls. This may include the LTPs (8 and 9) that were presented in the ExposureContext (17 and 18) by the ControlConstruct (19) representing the device functionality within the SDN Controller.

Depending upon the degree of mapping, the LTP in the network view may be identical to that presented in the subordinate ExposureContext of the device view and hence the same LTP instance can be aggregated by the ConstraintDomain associated with the ExposureContext of the ControlConstruct representing the SDN controller (3, 8 and 9) and the ConstraintDomain associated with the ExposureContext of the ControlConstruct representing the device (17 and 18). In the case depicted the LTPs are not identical ($8 \neq 17$ and $9 \neq 8$) and hence separate LTP instances are present (8 and 9).

In a more complex example, an LTP presented by one ControlConstruct may have two LPs but it is known that there are more LPs for the same LTP presented by another ControlConstruct. It is expected that a superior ControlConstruct will assemble (union) the fragments to form a coherent single entity using whatever matching criteria are appropriate. If a representation is a fragment, then appropriate match criteria and combination rules will need to be used to identify which fragments to combine to form the whole and what process to use to form the whole.

[4] The CIM should be used at all levels of view of networking capabilities. Clearly legacy devices will use traditional representation forms.

In a case where there is a simple consolidation of information it is possible to subsume the aggregated instances in several ConstraintDomains from subordinate ExposureContexts in a single ConstraintDomain of a superior ExposureContext so that there is a simple aggregation recursion. If the instances are identical, the ConstraintDomain of the superior ExposureContext can simple aggregate the same instances that are in the subordinate ExposureContext.

If a device is controlled via two ControlConstruct (along with other devices), each ControlConstruct will present the device as an ExposureContext, as noted above. Depending upon the specific realization, it is possible that the ConstraintDomains associated with both ExposureContext (one from each of the ControlConstructs) will have some entity instances in common.

As any entity instance can be represented in many views, the model accounts for controller resilience and control migration. A ControlConstruct can present the same information in several views. A ControlConstruct can present the same information through several ports.

Several different ControlConstructs can present the same information at the intersection of overlapping views. The UUID of the instance of an object presented in a view is provided by the ViewMappingFunction. Two distinct ViewMappingFunctions will provide different UUIDs for the abstraction from an underlying single entity instance. Specific properties, including IDs can be used to allow instance reconciliation[5].

Any representation of a thing in a view could be known to be a fragment (e.g., an FD could represent a fragment of the whole domain where forwarding is possible). This may be determined as a result of explicit or implicit off-network (out of view) relationships within the entity. It is expected that sufficient information will be provided to a superior controller that has a broad view to allow reconciliation and assembly of the fragments to form the whole instance.

4 Understanding the control component and view model

The world of networking has changed as computing and networking converge. It is clear that the implications are significant and there is an opportunity to take advantage of patterns that are apparent when taking a holistic view.

Traditionally Network Element, or a similar concept, has been used to represent a 'logical device'. This concept was easy to understand, especially when a 'device' had only one major function (like an SDH ADM or a PDH channel multiplexer).

As 'devices' have become more complex and multi-functional, the usefulness of the Network Element concept has decreased. For example, initially packet routers and Ethernet switches performed complementary functions. Now we have routers with inbuilt switches and layer2/3 switches, blurring the distinction between them.

[5] Each ControlConstruct instance has a distinct and different UUID but some of the object instances presented in one view may have the same UUID as object instances presented in another view as they are representations of the same thing. For example an LTP instance in one view may have a UUID of 27 and an LTP instance in another view may also be UUID 27.

Another point of confusion is where the management plane scope and the functional scope were mixed in concepts such as 'Managed Element' or 'Managed Network Element'. This scope confusion is especially problematic when 'devices' are logically partitioned or grouped to form 'distributed devices'.

The key to understanding the way forward is to understand that in a multi-functional 'device', we need to focus on the functions that the device performs. In hindsight, NetworkElement was just a container with equipment, that grew too complex and tried to encapsulate everything and ended up causing a lot of issues.

Reexamining the way of representing networking functionality leads to the Component-System pattern, the ProcessingConstruct and the approach to representation of control discussed in this document. In addition, the model of physical things set out in TR-512.6 cleanly separates genuinely physical things that can be measured with a ruler, from logical concepts. The general approach is careful separation of conceptually distinct concerns into functional, physical and informational and then to further separate functional into control and networking etc.

4.1 Rationale

The ONF Architecture [ONF TR-521] shows a recursion of control. This aligns with the ideas from [TMF IG1118] which:

- Developed the concept of the Management Control Continuum (MCC)
- Emphasized that automation is essentially about closing the control loop
- Explained the recursion of control loops where a control element may participate in one or more loops
- Developed the Component-System pattern
- Emphasized that a Component exposes views
- Explained how a ControlConstruct exposed views of itself and what it is controlling to its client (which were potentially simply control components with broader scope)
- Highlighted recursive functional abstractions, where a selection of functional components offered by providers are taken by a client, pruned to give useful function, assembled into a system and the capabilities of that system are offered to clients in various pruned and refactored functional component forms. Offered functional components are then taken by a client and the process is repeated.
- Explained that all functional capabilities viewed are abstractions of an underlying system with greater detail and complexity, and are, as a consequence, also virtualized within the scope of the provider system.

An SDN Controller will be realized using compute, storage and communications capabilities. Clearly the traditional SDN Controller just like the traditional Network/Managed Element will have communication ports. These communication ports have functionality that is no different from any other function terminating a stream of packets. The functions of communication ports of the SDN Controller are represented using the LTP class. Hence a control device and a transport NE are essentially the same. All such devices are balances of compute, storage and communications capabilities (it is just the specific balance that is different).

4.2 Implications

Three classes from the V1.2 model are obsoleted and replaced:

- SdnController of V1.2 becomes a ControlConstruct
- NetworkControlDomain of V1.2 becomes an ExposureContext of a ControlConstruct that represents the SDN Controller
- NetworkElement of V1.2 becomes one or more ExposureContext for the ControlConstruct of the device where each has been mapped to an appropriate exposure, representing
    - The view of the capability of the ControlConstruct itself in the device.
    - The view of the key network etc. functions (i.e., the LTPs etc.) related to the ControlConstruct in the device

The relationship between the ExposureContext and the things in the view is aggregation within a related ConstraintDomain and not direct composition as it was in a traditional model of an NE.

Where there is an embedded control plane/controller that is essentially independent of the NetworkElement, this can also be represented by a ControlConstruct and one or more ExposureContexts.

If there is an opportunity to see the native model of the NE as well as the mapped model then an the ControlConstruct that represents the NetworkElement will also have a ExposureContext exposing device specific classes. In this case, it would be expected that the ControlConstruct that represents the device would make available the ViewMappingFunctions that "explain" the relationship between the views provided.

We can use ExposureContext and its associated ConstraintDomain to represent:

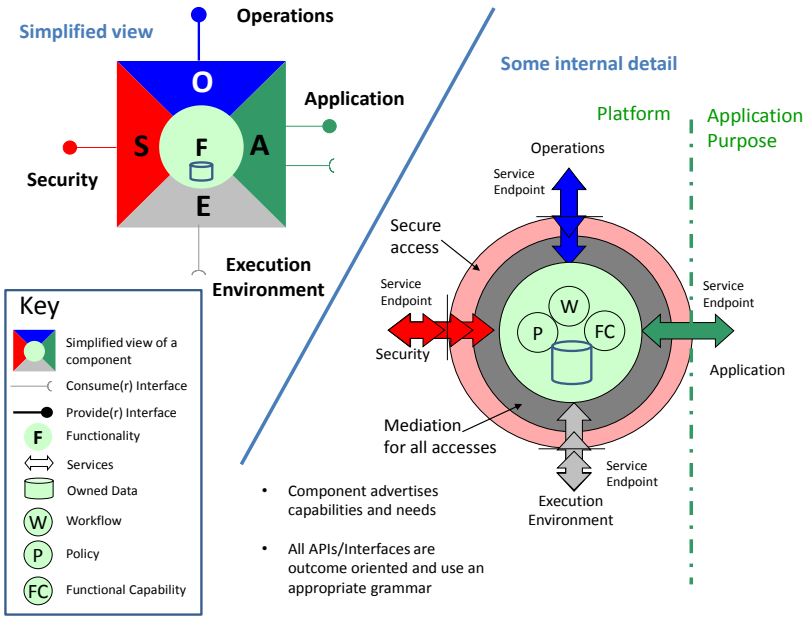
- A logical scope that aligns to a physical inventory boundary (especially useful for 'device partitions' and 'distributed devices')
- A management scope (which may differ from the physical and functional scope)
- A general functional scope that can be used for grouping and scope boundaries

While the move to replace NetworkElement with ControlConstruct and ExposureContext was prompted by issues in representing 'traditional devices', it can be seen that (along with the existing decoupling of functional and physical viewpoints) this now gives a neat and consistent representation of SDN and NFV implementations, where the NetworkElement concept is largely irrelevant anyway.

4.3 The patterns behind the model

As for all components, the ControlConstruct has ports. The ports provide access to the ControlViews and allow control of the ControlConstruct.

A helpful view of this is provided by [TMF IG1118] as shown below.



[TMF IG11118] Figure 1 The FMO component interface and structural overview

Figure 4-1 A Controllable Component

A Component has an Operations port through which it may be controlled/managed[6] and an Application port through which it exposes it purposeful behavior. The purposeful behavior of a Control Component is related to the controlling of other Components, A Control Component has an Operations port through which it is controlled.

As discussed in TR-512.A.2, all functional capabilities of the network are represented in the form of Components (FC, LTP, PC etc.). Likewise, the functional capabilities of the control system can be represented in the form of Components (e.g. C&SC).

The ports of the control components used for signaling can be represented using LTPs and the Control Functions that terminate the signaling can be represented by Control Components such as C&SC. Where appropriate, the signaling itself can be represented via a protocol definition perhaps using the Generalized operations pattern (see TR-512.10).

4.4 Identifiers, naming and addressing

In general, there is a need for separate spaces of identifiers/addressing for:

- Ports
- Control functions
- Management-Control views (including virtual views)
- Functions (Virtual)
- Physical things
- Mixed assemblies of Functions and Physical things
- Places

[6] A component provides a façade through which it can be controlled…. This essentially provides access to an embedded controller which is at the lowest level of "visible" recursion (degenerates to a transistor gate etc).

- Reference points (e.g., UNI, or at a named API)
- Resources (compute, networking, storage)

When a controlled thing does not have a native UUID that can be used consistently across Control Views, there needs to be some directory service to provide consistent identification. An example of a directory service is the ITU-T G.7701 directory service component.

4.5 Resilience in the Control System

By separating the identifier spaces for Control from the spaces of the things being controlled and by loosening the association from composition in a traditional model to aggregation, the Control model is then set up appropriately to allow for well identified instances of controlled things to appear in more than one ControlView. As a consequence, various controller resilience schemes are readily supported. The client contexts of the ONF Architecture [ONF TR-521] would hold name spaces that could point to shared resources between client contexts.

4.6 Controller view considerations

The figure below highlights the pattern of talking through a port to a controller about a controlled system where that system:

- Includes the controller itself
- Is represented in terms of components
- Is represented via some pruning & refactoring transform

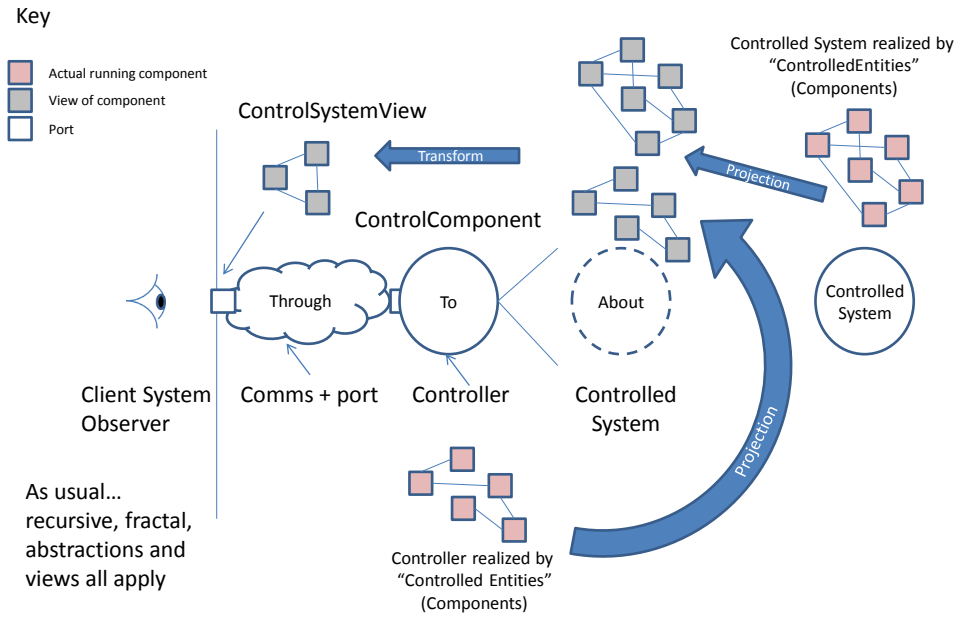


Figure 4-2 Through, To, About…

The figure below shows the perception of a complex network as viewed by the Client. The ExposureContext, via the associated ConstraintDomain, will include precisely the functional components perceived by the Client. The perceived functions are an abstraction of the actual network and are also virtualized in that the Client does not know, or care, where the functions actually are. The figure shows a network that has a function "B" that is exposed as "Func B' " to the Client.

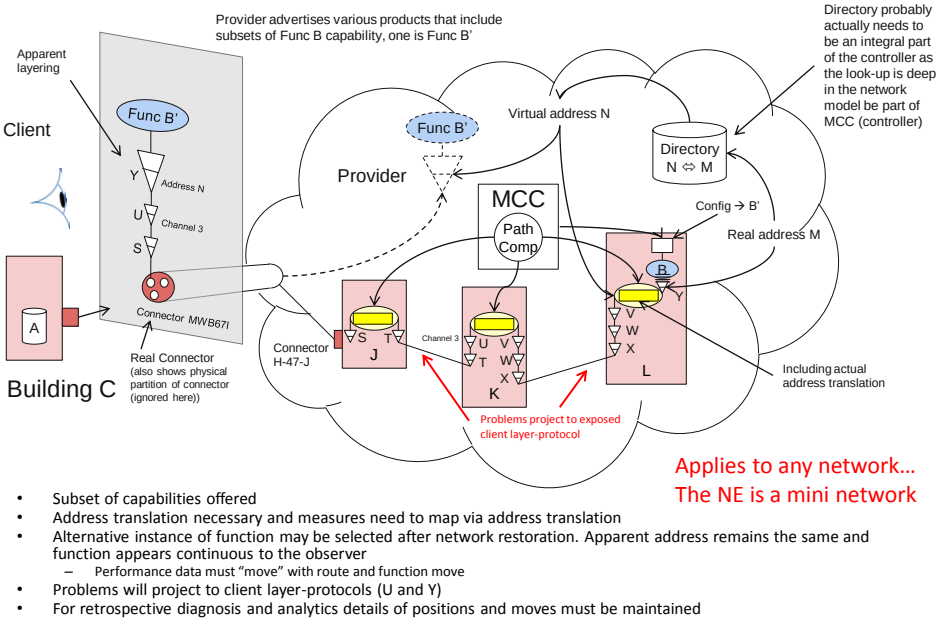


Figure 4-3 Simple network view mapping

The figure below shows a network that has a virtual function "B" (virtual) that is exposed as "Func B' " to the Client. The view provided to the client is the same as in the previous figure although the realization in the network is quite different

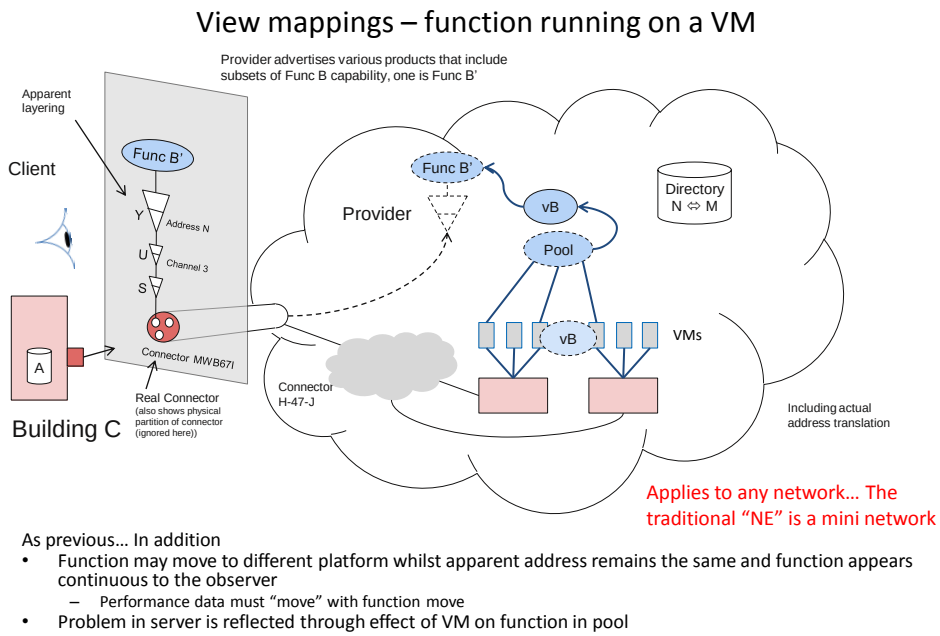


Figure 4-4 View mapping for functions on a VM

The figure below shows a client view of various control interfaces related to a particular simple network service. The same pattern applies at all levels and as a consequence the same model can be applied at all levels. Traditionally different models have been applied.

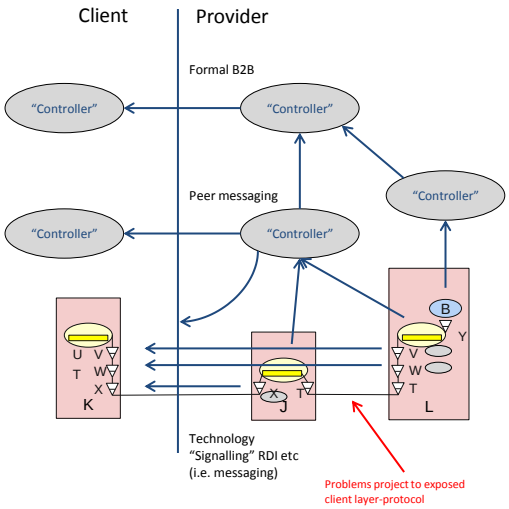


Figure 4-5 Client view of network and control

The diagram above highlights the following:

- Signalling is messaging
- Network device essentially has embedded controller

    - The embedded controller generates messaging at the "network technology level" (traditionally called signalling)
- Messaging at the network technology level is "immediate" but provides minimal information and hence causes somewhat "knee-jerk" actions
- Higher controller provides richer information but with reduced immediacy
- Higher controller may drive network technology level messaging (signalling)
- In the longer term embedded controller become part of the continuum
- Approach to messaging source depends upon trust and information usage

The figure below shows a simplified picture of the client view of an actual service (capability) and view of control of that capability. The figure uses the symbol set highlighted earlier in this section from [TMF IG1118]

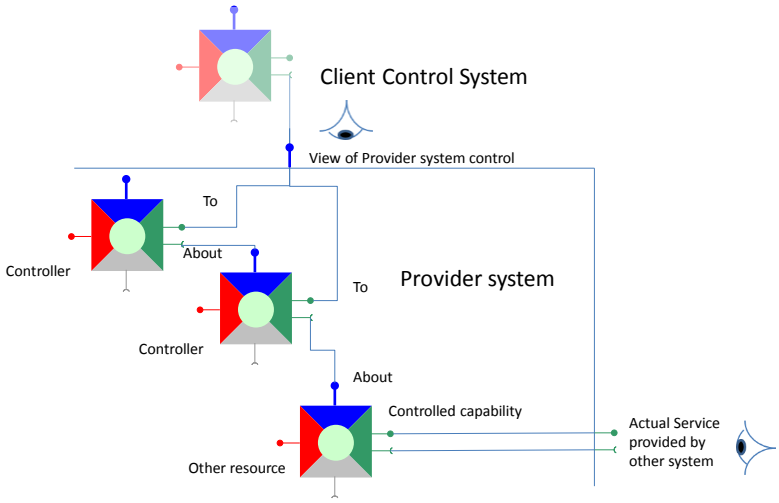


Figure 4-6 Simplified view showing exposure of controllable capability to a client

4.7 Dismantling the NE – Some rationale

The Network Element (NE) concept has been around for a long time.

- A Network Element is defined in US law[7] as "Network element is defined as a facility or equipment used to provide a telecommunications service. Such term also includes features, functions, and capabilities that are provided by means of such facility or equipment, including subscriber numbers, databases, signalling systems, and information sufficient for billing and collection, or used in the transmission, routing, or other provision of a telecommunications"
- [ITU-T Q.1741.9] defines NetworkElement as "A discrete telecommunications entity, which can be managed over a specific interface, e.g., the RNC."
- [ITU-T G.780] defines "network element (NE)" as "A stand-alone physical entity that supports at least network element functions (NEFs) …"

[7] https://definitions.uslegal.com/n/network-element/

The NE is a somewhat messy thing. One of the issues we have is that existing representations make a number of assumptions that aren't true in many cases. To avoid confusion by redefining the existing concepts, new terms are required to clearly define what it is and isn't.

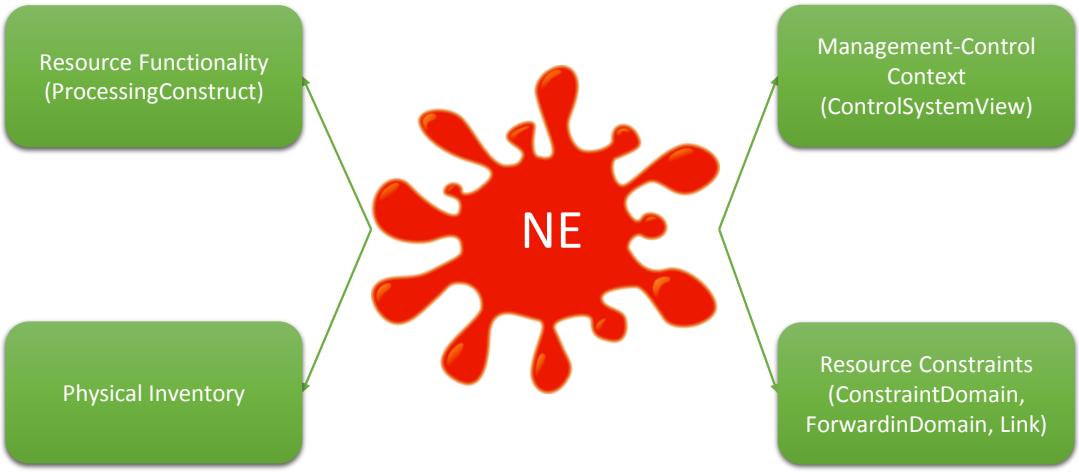


Figure 4-7 The "NE"

A much cleaner, recursive and consistent model has been formulated that takes advantage of the Control-View model discussed above.

The following section discusses the rational for dismantling of the NE.

4.7.1 The analysis

Looking broadly at the drivers from earlier sections:

- The Management-Control Continuum, as identified by TM Forum, extends down through the SDN Controller into the NE such that an aspect of the NE is a controller
    - The SDN Controller looks like any other manager/controller
    - The NE looks, in part, like any other manager/controller
- A generalized model of control, access to control and control scope will provide a basis for a coherent reworking of both the NE and SDN controller representation
- The SDN Controller, like the NE, needs to present a representation of the functionality it is controlling as well as to present itself as a set of control functions
- There appears to be a need for a generalized representation (pattern) of a coherent unit of functionality
    - To cover both control functions and controlled functions
- Just as for the NE, there needs to be a representation of the relationship between the function (of control and being controlled) and their physical realization
    - The representation of physical realization using the Equipment model will bring geographical positioning information
        - The control/communications channels for both the NE and the SDN Controller look like any other communications

        - The representation of communication channels using FC/LTP will link with the remainder of the communications network
    - The relationship between a function and its physical realization may be through many levels of functional realization

The Network Element (NE), as concept, is a somewhat incoherent hybrid of various concerns where the hybrid is not viable for many cases. One aspect of the NE is control and this should be represented and considered in the same way as any other controller. The control aspect is the primary focus of a Managed Element (ME) but this also suffers from the same lack of coherence.

Clarity is brought by considering the separable concerns:

- Physical thing (solid i.e., a thing that can be measured with a ruler and has weight) and Physical space (i.e., with volume but no relevant weight)
    - A coherent physical thing that in context is not relevantly decomposable (component, atomic)
    - A coherent assembly of physical things (system/assembly, composite)
    - Similarly physical space
- Positioning of the physical thing in geographical space
    - Essentially a point in space (very small geographical area)
    - A large geographical area
- Virtual[8] function emergent from a physical thing where the virtual function has capability and is potentially active
    - A coherent virtual thing that is in context not relevantly decomposable (component, atomic)
    - A coherent assembly of virtual things (system/assembly, composite)
    - Only realisable via supporting physical things (see TR-512.6 for details of the relationship between the models of physical and functional things).
- Management-Control function, Management-Control scope and access to Manage-Control where that Management-Control function
    - The functions that fulfil and assure the intent and that provide access (can be talked to) to a view of things (that can be talked about)
    - Is itself a virtual function
    - Can view and manipulate virtual functions
    - Can provide a view of Physical things through virtual functions
- Port through which to access management-control information
    - Will necessarily be a partial view of information of each thing that can be viewed
    - May overlap with the view provided via another management access (such that some things are seen partly through one port, partly through another and partly through both)
    - May allow access to information on geographically distributes things
    - May allow access to information representing fragments of functionality some of which may be completely disjoint from others
- A named hybrid assembly of virtual and physical things spread over an arbitrary geographical area

[8] Also called Logical Function.

- The assembly of information that can be accessed through a management port

The NE is a mix of the above (as is the SDN Controller, the EMS etc.). The challenges with the above conglomeration approach:

- Inconsistent boundaries
    - The boundary of a coherent physical thing is highly unlikely to be coincident with a coherent virtual thing
    - The boundary of the visibility via the management access is likely to cut across the boundary of physical and virtual things
    - Some disjoint things are accessible via the same management access
- Geographical spread
    - The management access may be to things that are spread across geography and hence:
        - Themselves do not have shared fate
        - Have shared fate with things accessible via other management accesses
- Identity and name challenge
    - Each instance of the concept has identity and some form of identifier in a context that allows identification and potentially allows location via some form of address
    - The identifier for the management access may differ from the identifier for the various virtual things and for the various physical things accessible
    - The same thing may be accessed via management accesses of several different controllers
    - Accidental use of the same identifier for multiple purposes (e.g., a function, and a point on a piece of equipment) with no clear name space separation
- Lifecycle fragmentation
    - A virtual thing visible via the management access may persist beyond the life of the management access etc.
- The assembly of information that can be accessed through a management port
    - For a geographically distributed "ME/NE" it is potentially necessary to open up the "ME/NE" to understand its cabling etc. and fate share with other systems
    - An "ME/NE" may group multiple "subnetworks" and have internal interconnecting "links"
- A composite "ME/NE" may provide access to disjoint functions that have independent network purpose
    - For example, an FRU that only draws power and perhaps receives basic control and that has no functions relevant to the rest the FRUs in a shelf that forms part of an NE
- Some things may be accessible as if in two different "MEs"/NEs"

Considering the current model clearly physical and functional things can be represented. Hence the focus of the model to replace the NE is the control view and the control entities themselves (the control entities are controllable).

- The control entities can be considered as Components.
- In a controller view, there is potentially a view of the view provided by the subordinate controller (and so on)

    - The critical consideration is what needs to be exposed. The "NE" exposes a view. The controller of the NE "may choose" to expose a view which may include the NE view or an abstraction of it (which the controller may claim is the NE view
- A view is accessible through a port and a port is an LTP (which is a component-system)
    - There is an address of the port at which the information the controller expose is available

All systems involved in Control (e.g., NE, EMS, NMS, SDN Controller, Orchestrators) can be treated in the same way.

- The views are aggregation. The provider of the view can be removed without the system ceasing to function
    - The lifecycle of the presentation is independent of the lifecycle of the presenter
    - A view may be provided through several accesses. An LTP could be visible through multiple views
    - There could be fragments of entities provided in a view where the whole entity is made by assembling information from several views
    - It is the ControlEntity that is requested to perform actions on the things presented through the view
- NE cases illustrating points on the broad spectrum
    - A simple regenerator which is a single piece of hardware with one function and two… this is clearly representable as a traditional NE (single geographical place etc.)
    - The DSL case with a direct access to the remote and a head end that consolidates the remote. If monolithic NEs are considered then there is a problem, if views are considered then there is no problem.
- Control of a "white box" NE will benefit from this approach
    - The views are decoupled from the physical platform and from the ControlEntity. They can move. The location of the producer of the view is determined via the relationships to the equipment model.
        - Equipment gives rise to function gives rise to complex function gives rise to LTP
- There is no need to create a virtual NE or virtual hardware.
    - Simple view based or domain based groupings of functionality covers all cases

The following figure shows an NE that happens to be significantly geographically distributed.

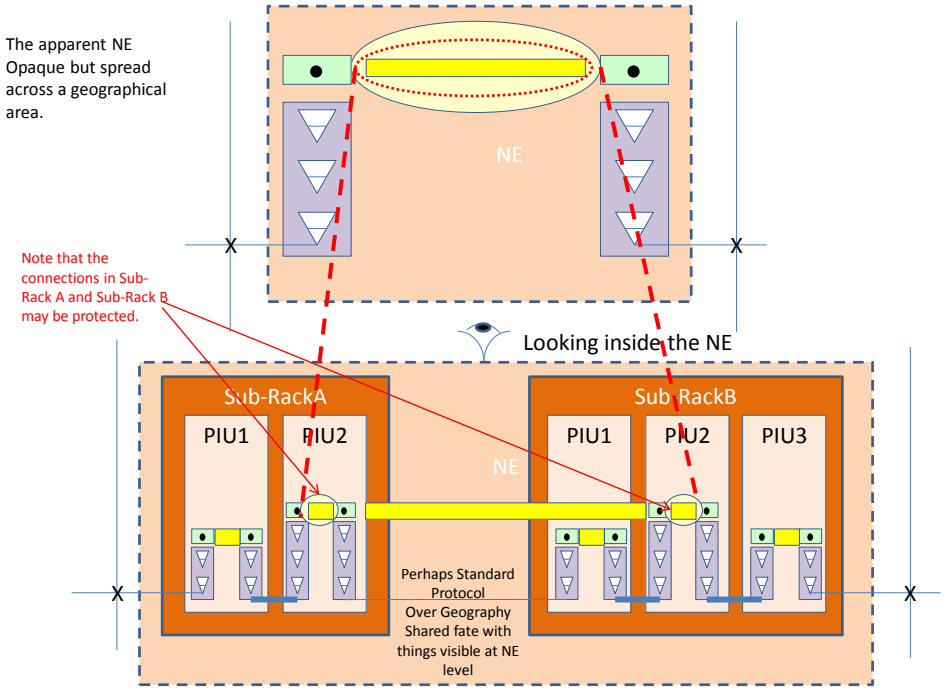


Figure 4-8 Geographically distributed NE

In the figure above:

- A subset of functions form a coherent unit of stand-alone network function
- There is significant geographical distance between two functions accessible through the control interface

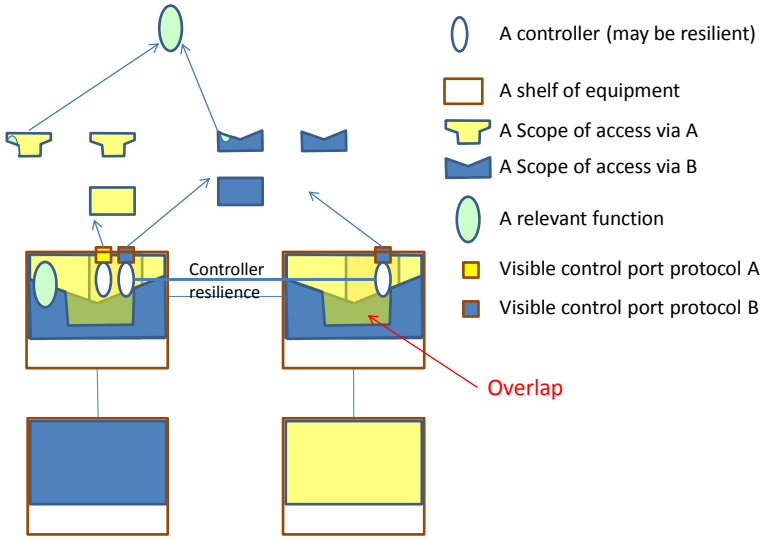


Figure 4-9 An NE with two control access ports each providing a partial view

In the figure above, an assembly of equipment forms a traditional NE that happens to have two control access ports, each providing a partial view. Part of a relevant function (e.g., an LTP) is accessible through one control interface and part through another.

4.8 The control model applied to the "Controller"

The control model discussed here can be applied to any manager/orchestrator/controller. The ControlConstruct can be used to represent any control functions. If a more detailed functional model of the Controller is required, the model described in this document can be supplemented with the ProcessingConstruct/ConstraintDomain (see TR-512.11). The Controller model is not fully developed in this release.

4.9 The configurationAndSwitchController (C&SC)

The C&SC is described in TR-512.5. It is a specialized ControlConstruct used for control of forwarding resilience. It is expected that the C&SC, the Control model described here and the PC/CD model will be further refined in subsequent releases.

5 Operations

This section considers the details of exposing capability on a ControlPort. This section describes a generalized model of interfacing. In a real implementation, if the ControlConstruct was representing client facing capability of an SDN Controller, then the ControlPort may be exposing a TAPI interface.

5.1 The basic model

The following figure shows that a ControlPort can take either a ProviderRole or a UserRole.

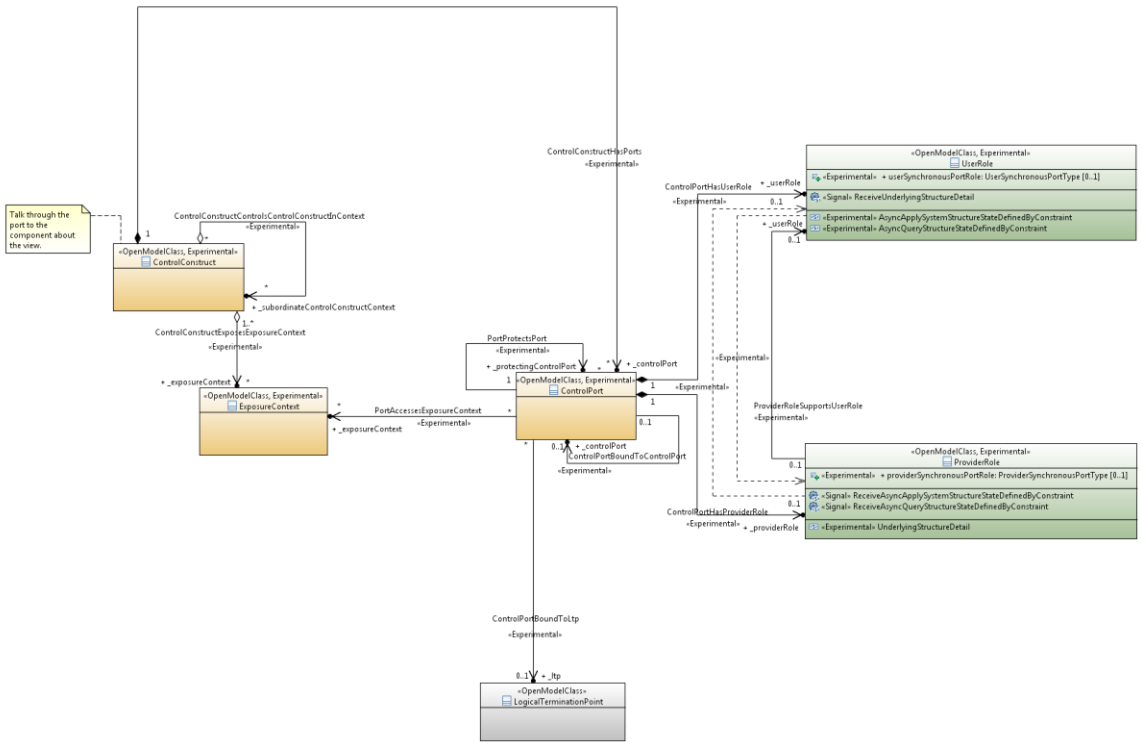
The ProviderRole offers:

- A synchronous interaction opportunity for traditional message/response interaction covering both:
  - Request for information
  - Request for change to be made[9]
- A notification opportunity to enable reporting of changes in state of underlying structure detail covering:
  - Changes in the controlled resources, e.g., the network
  - Changes in the control system (also therefore controlled resources)
- Two signal receipt opportunities for use in an event driven context where the signals, potentially broadcast by the client, are related to request for information and request for change

The UserRole offers the other half of each of the ProviderRole opportunities.

The two information flows (dashed lines) represent Provider to User and User to Provider flows.

The detail of the model is covered in later figures.



CoreModel diagram: Control-ControlPortWithProviderAndUserRole

Figure 5-1 Provider and User role ControlPorts

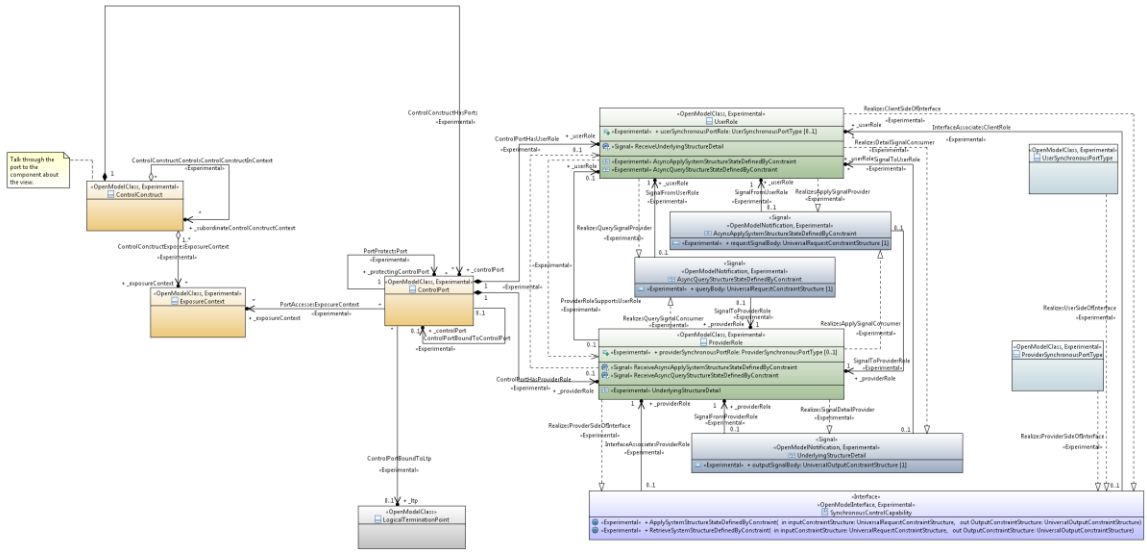
[9] Change may change a value, cause linear behaviour, cause some rate of change etc.

5.2 Provider and User role detail

This section expands on the model introduced in the previous section and introduces some more detail on the structures in the previous section.

In the figure below the three signals are expanded and the notion of universal structures is highlighted. These structures are further explored in later sections.

In addition the interface that is formed by binding a Provider role ControlPort and User role ControlPort is depicted (in violet). This also uses the two universal structures.



CoreModel diagram: Control-ControlConstructShowingProviderAndUserDetail

Figure 5-2 Provider and User role detail

5.3 Long-lived operations and Universal structures

A request for change, conveyed either via the synchronous or asynchronous mechanisms highlighted above, is expressed in terms of the UniversalRequestConstraintStructure which is a statement of desired outcome and is in the form of the OperationEnvelope described in TR-512.10.

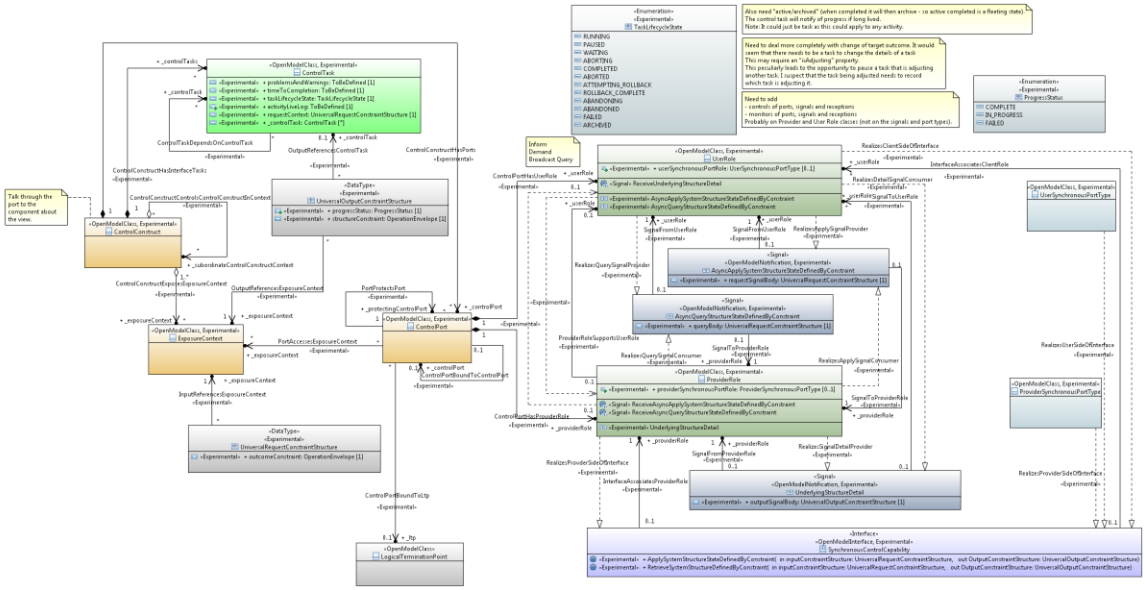
A synchronous interaction is reasonable when the provider task is simple such that the provider can easily provide a response in a timeframe considered suitable by the user. In this case the response may be COMPLETE or FAILED along with full relevant details in the structureConstraint attribute along with pertinent notifications from the network resources etc represented in the ExposureContext.

In cases where the task duration is long or is widely variable, initiation via a synchronous interaction is still possible but it is then necessary to provide a rapid response followed by updates on progress up to completion. The following figure introduces the ControlTask which will provide visibility of the progression of the task. This is highlighted to the client via the UniversalOutputConstraint Structure supplied in the response to the request which also supplies the ExposureContext reference.

This model fragment allows a Controller to respond with either the complete answer, in a structureConstraint or with a partial answer (potentially simply IN_PROGRESS) along with a reference to a ControlTask entity that can be queried for progress of the task and that will notify of changes in the task. The ControlTask provides the full request detail via the requestContext property.

Where the task has not been completed the relevant ControlPort will emit notifications related to progress of the task in terms of

- Entities created etc, such as FCs, LTPs etc (which may be collected together into sugraph assemblies)
- ControlTask state and property changes where the ControlTask can progress through the any relevant TaskLifecycleStates.



CoreModel diagram: Control-ControlConstructWithUniversalStructures

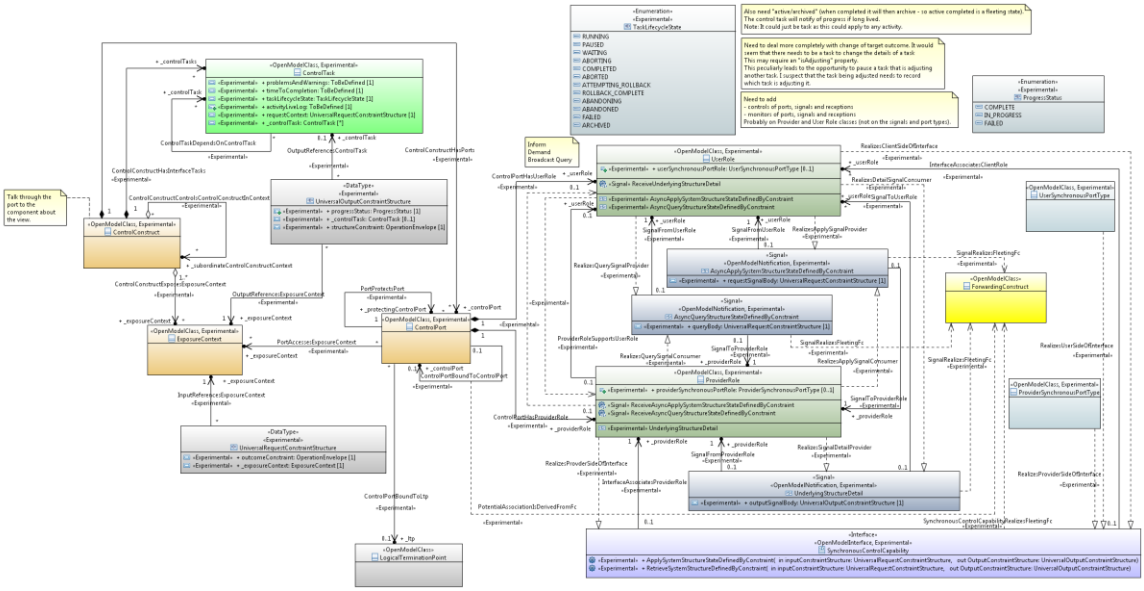
Figure 5-3 Long lived operations and universal structures

5.4 The full model

The ControlPort is supported by an LTP which will encode and propagate the relevant messages. The LTP will have associated to it FCs that provide the necessary connectivity to enable communication.

Depending upon the protocol there may be a session running between communicating ControlPorts. This session can be represented by an FC between LTPs with the appropriate LayerProtocol.

Where there is no session the momentary relationship made as a message leaves on ControlPort and arrives at another ControlConstruct port can be considered as a fleeting FC. Clearly, it is unlikely that instances of this FC will need to be modelled in any way.



CoreModel diagram: Control-ControlConstructWithFullOperationsModel

Figure 5-4 Full Control Model

End of document