Core Information Model (CoreModel)

TR-512.8

Control

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	Major rework of model.
1.5	September 2021	Enhancements to model structure.
1.6	January 2024	Addition of streaming model and ControlTask model detail.

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 the Control model

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.

As explained in [ONF TR-512 V1.2] the classes SdnController[1], NetworkControlDomain and NetworkElement[2] have been reassessed and deprecated. New classes were developed in release V1.3 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.

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

[1] In general, a controller, or control system, is a collection of functions that are designed to act on another system (the controlled system) for the purpose of controlling that system where the act of controlling is intended to manipulate the state of the controlled system such that its behaviour is as defined by some user of that controlled system. As a consequence, the controller must monitor the state of the controlled system and act to maintain the desired behaviour. To do this there will necessarily be some feedback loop realized in the controller.

[2] 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 [ITU-T G.8152.1] (MPLS-TP).

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 V1.1] 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 43 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 by 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 (MCC) – see section 4.1 Rationale on page 35). 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

✓ = dominant function
- = 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 significantly 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.
- Whilst the figure shows only CDs bounded by a physical chassis, this is a diagrammatic simplification. In general, a "network function" and hence a CD can be distributed across multiple chassis (and a physical chassis can support multiple CDs some of which may be distributed across multiple chassis).

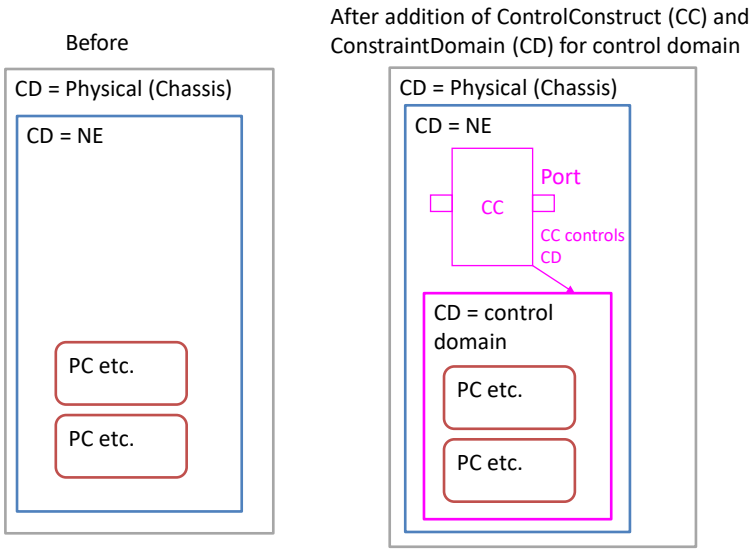
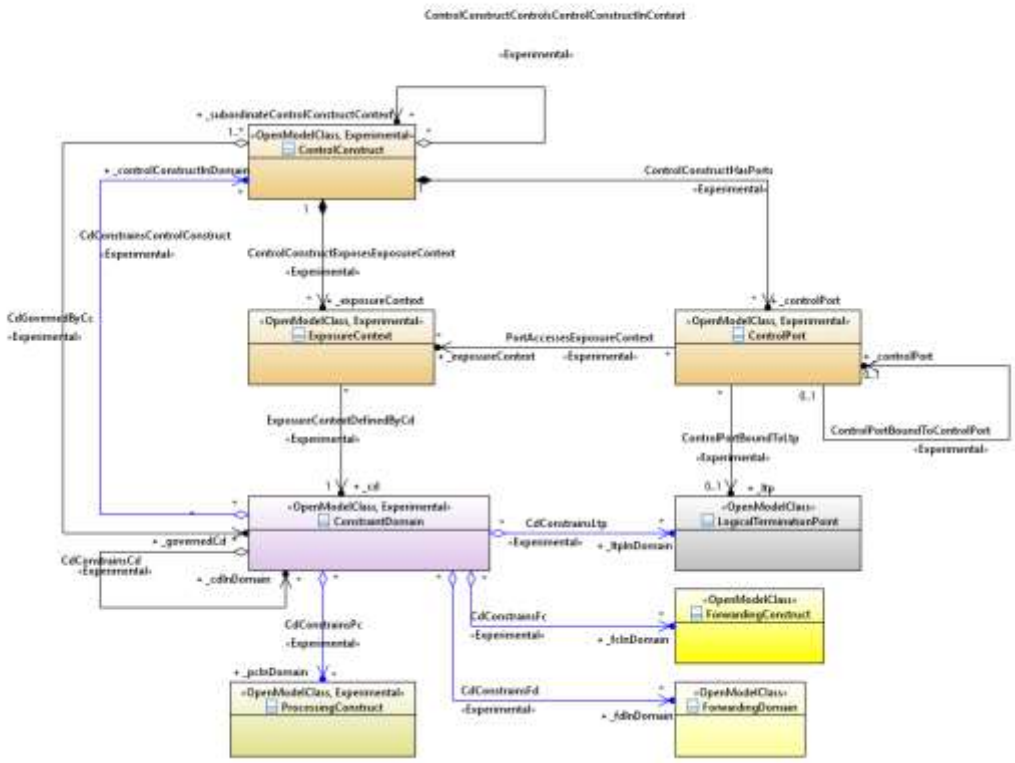


Figure 3-2 – Basic Network Element

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 a 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.

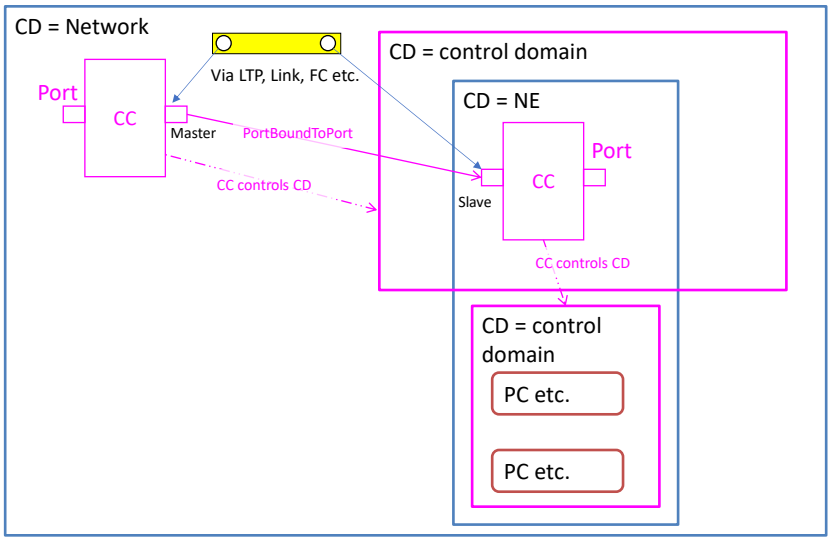


Figure 3-4 - Basic ControlConstruct layering Use Case

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 are no associated transport requirements and hence these bindings can be omitted in an implementation, reducing the complexity of the information stored.

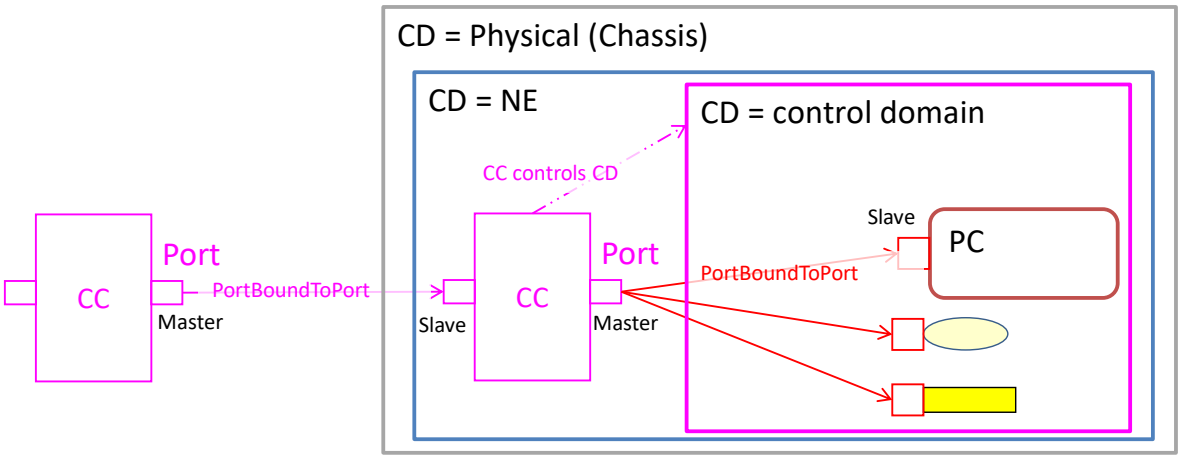


Figure 3-5 - Control port to "PC etc." port binding

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 but:

- 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 and ControlConstructs. 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.

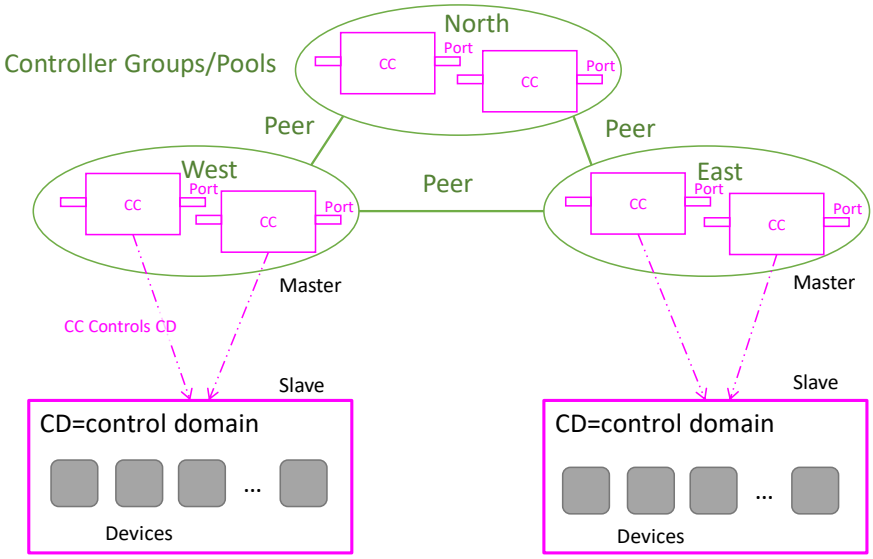


Figure 3-6 - A mix of Master-Slave and Peering

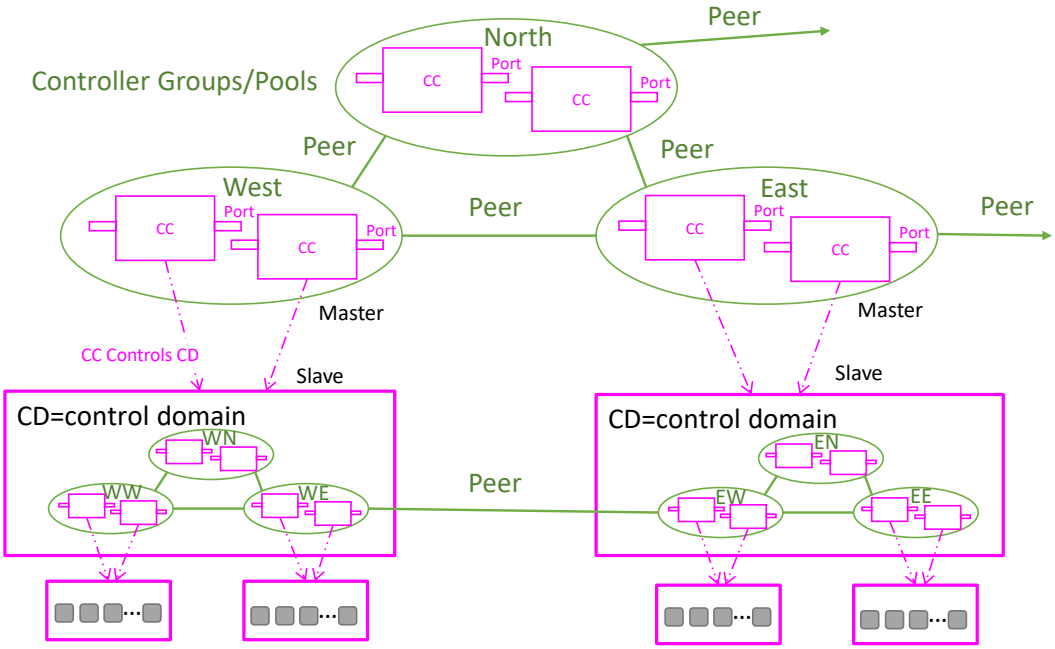


Figure 3-7 - Recursive Control Architecture

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. The behavior at the receiving Control, which involves a receive side ExposureContext, is discussed in section 7.6.3 Stream client on page 92.

Note that an Exposure Session (formed between the client and provider via their ControlPorts and related to a particular ExposureContext) can be considered to be a form of secure 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

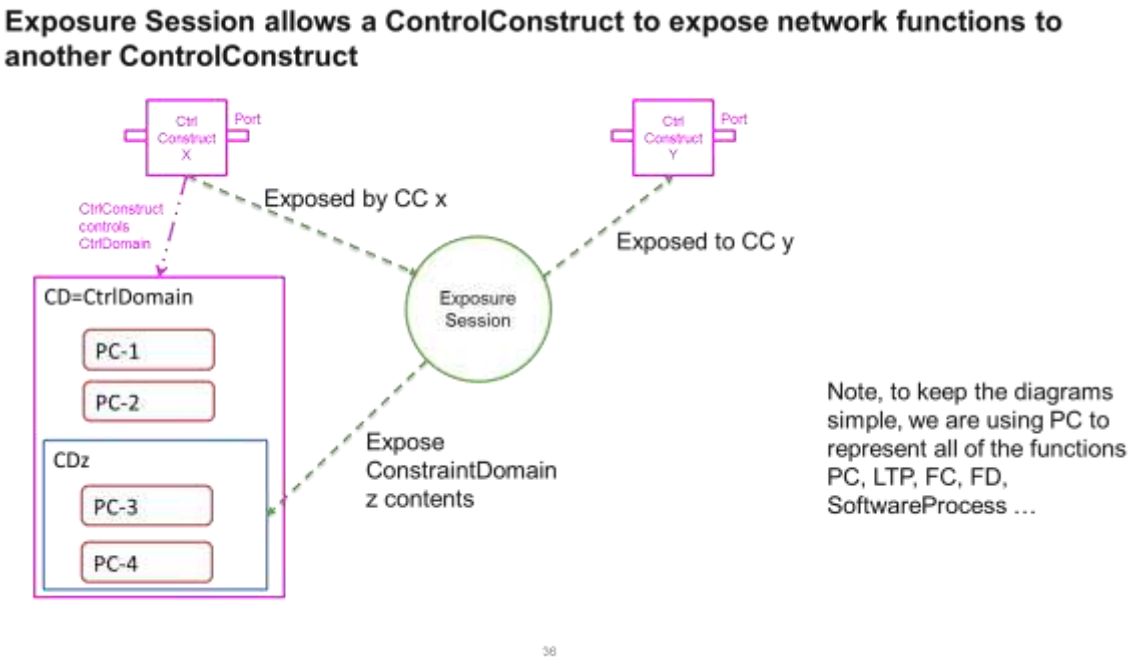


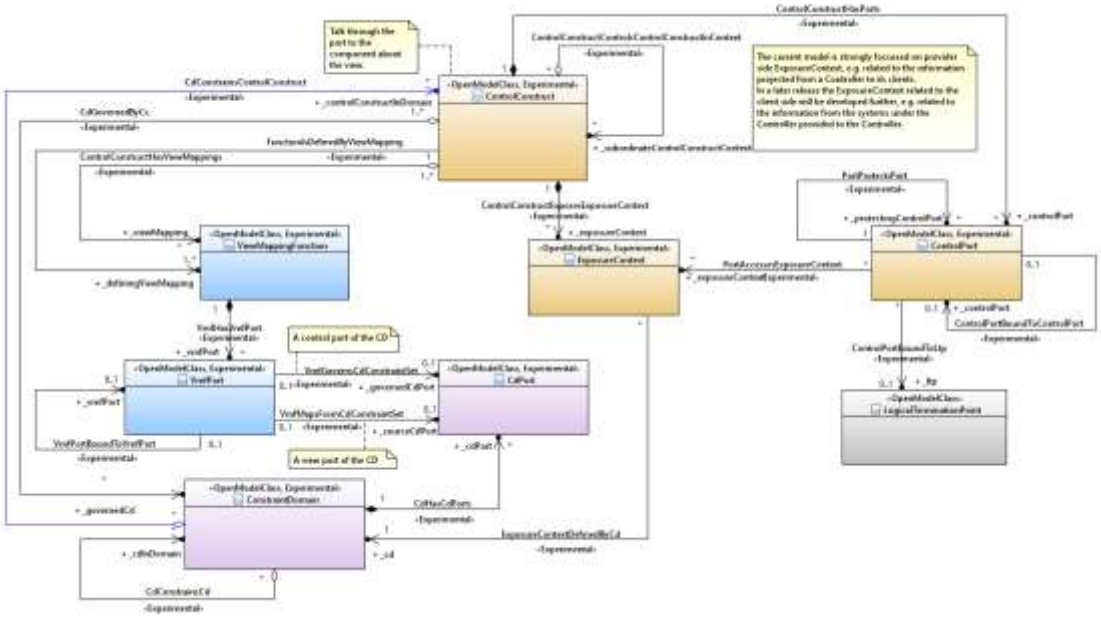
Figure 3-8 – Exposure Session[3]

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 core

The figure below shows the core of the Control model.

[3] The figure shows exposure session. The figures in this section focus on the provider side modeling and do not show access to the constraint domain on the client side. They do not show how information gets from the ControlPort to the ConstraintDomain (which represents the image of the provider side). See section 7.6.3 Stream client on page 67.



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::ControlConstruct::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.

Inherits properties from:

- GlobalClass

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.
_requestConstructor		See referenced class
_receiveStreamPipelineBuffer		See referenced class
_logStreamControl	Experimental	See referenced class
_transmitStreamPipeline	Experimental	See referenced class
_governedCd	Experimental	A Constraint Domain governed by the Control Construct. A Constraint Domain may be governed by more than one Control Construct (shared) during a handover. This association is a lifecycle aggregate where the Constraint Domain must be governed by at least one Control Construct. A Control Construct may modify and delete a Constraint Domain it governs. A Control Construct may create a Constraint Domain. It is not expected that a Control Construct will govern a Constraint Domain that constrains the governing Control Construct.

3.3.2 ControlPort[4]

Qualified Name: CoreModel::GeneralControllerModel::ControlConstruct::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 be bidirectional.
This class is Experimental.

[4] A ControlPort instance would expose information about the capabilities that can be accessed through that ControlPort and about the methods available to access those, i.e., it would describe protocols, interaction methods, task opportunities etc. This has not been covered by this document in this release.

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.
_userRole	Experimental	Properties relevant when the ControlPort is exposing the ControlConstruct as a user of capability.
_exposureContext	Experimental	A view presented through the ControlPort.
_receiveFilterResponse		Passes the response to a previously made request to the receive filter for onward directing.
_receiveStreamPipelineBuffer		See referenced class
_receiveFilterRequest		Passes a request to the receive filter for onward directing.
_streamServer	Experimental	See referenced class

3.3.3 ExposureContext[5,6]

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

Exposes a view of the things controlled by a control system. For example of resources defined in this model (and referenced by clause A.10 of ONF TR-521).
The referenced ConstraintDomain bounds the view which is a structured presentation of the underlying controlled things (the "actual" entities) for some purpose.
The ExposureContext provides access to the view.
It may further constrain the capabilities supported by the view (e.g., read only).
It does not provide a different view as its only source of information is the associated CD and the purpose of the ExposureContext is to expose the view as provided by the CD.

[5] 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.
[6] More than one EC per CD is allowed, but each EC only deals with one CD. This lets us do some filtering in the EC (although the Transmit filter could do that too). The EC could prune content out of the CD, but it does not transform. So potential for a subset, although the main usage would be the whole set.

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.
_controlPort		See referenced class
_TransmitFilter	Experimental	See referenced class

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 (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 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 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.
_runningOsInDomain	Experimental	A RunningOs constrained by the ConstraintDomain.
_runningSoftwareApplicationInDomain	Experimental	A RunningSoftwareApplication constrained by the ConstraintDomain.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_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

3.3.5 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::ViewMappingFunction::ViewMappingFunction

The rules that relate one view to another and enable the transformation from one view to another.
A ControlConstruct aggregates ViewMappingFunctions.
The ViewMappingFunction is applied to the entities aggregated by one or more ConstraintDomains (via VmfPort - CdPort VmfMapsFromCdConstraintSet association) 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
_vmfPort	Experimental	A port of the ViewMappingFunction.

3.3.7 VmfPort

Qualified Name: CoreModel::GeneralControllerModel::ViewMappingFunction::VmfPort

A port of the MappingFunction.
This can provide an input to the mapping or an output from 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 expressed 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 ControlPort instances. Several ControlPort 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[7] 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 (as there is a need for a view mapping process to bring together several inputs to form an output). 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.

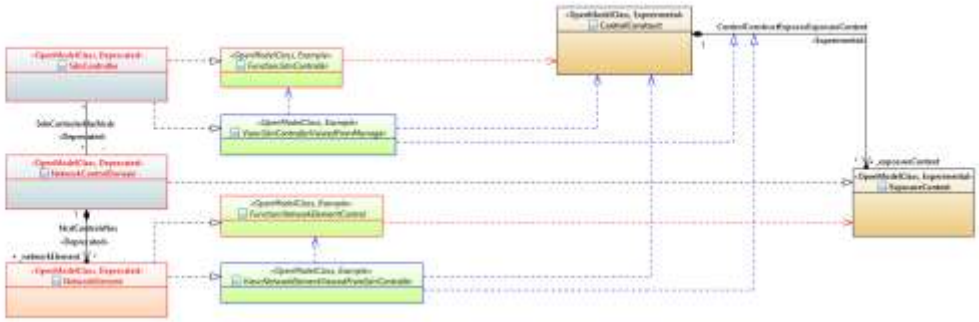
For example, a "Network Element" may have a property recorded against a port where that property was extracted from an incoming signal and where that property is defined as some form of discovery tag which, unknown to the NE has been sent by another NE. As the overarching controller can see both NEs (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 (see example in 3.8 General discussion on page 31).

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

[7] Same in that it has the same entities and properties as provided by the CD. The ECs restrict degree of access, such as read only, and feed a specific set of partitions of view.

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[8]
- 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)

[8] 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.

    - 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:

- 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::ControlDiagrams::mappingToTraditionalModel::explanatoryModel::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::ControlDiagrams::mappingToTraditionalModel::explanatoryModel::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::ControlDiagrams::mappingToTraditionalModel::explanatoryModel::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

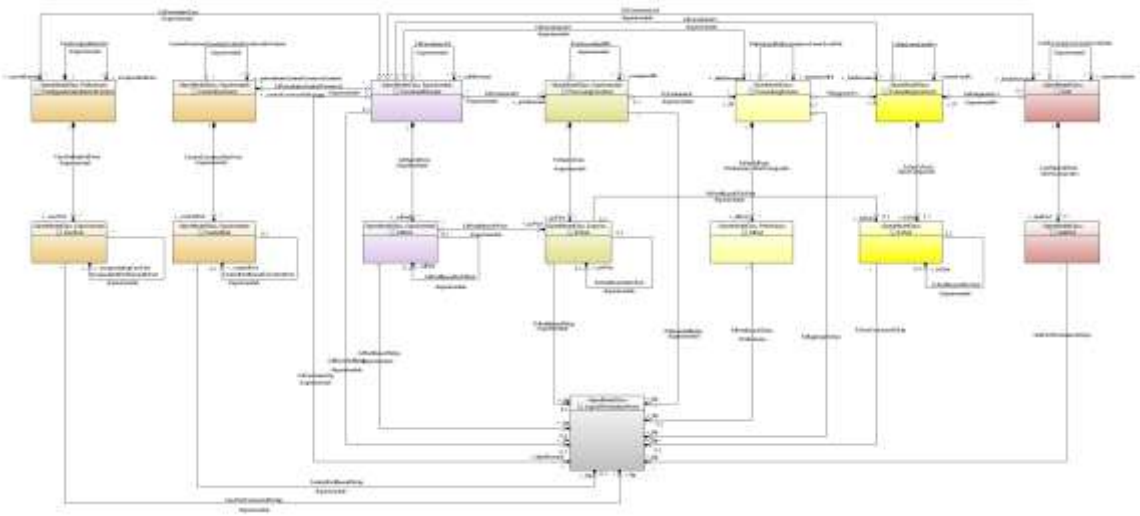
Qualified Name:
CoreModel::GeneralControllerModel::ControlDiagrams::mappingToTraditionalModel::explanatoryModel::View:SdnControllerViewedFromManager

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.

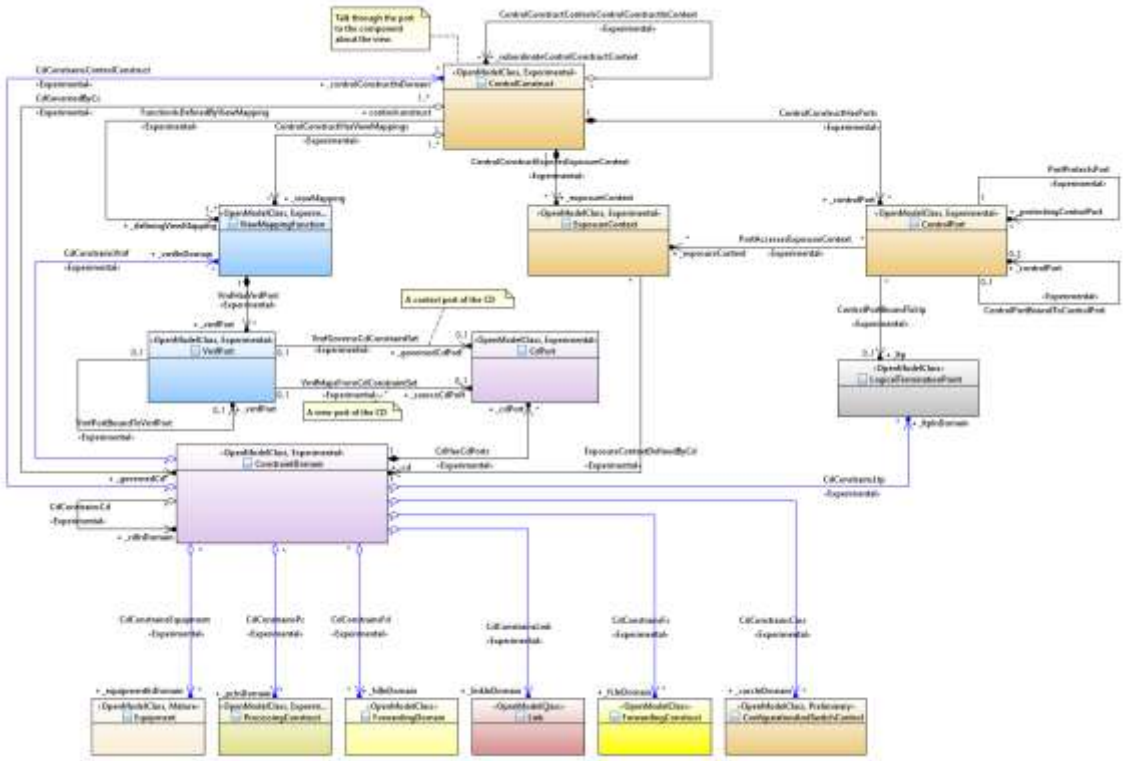


CoreModel diagram: Control-ControlConstructPattern

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 (CASC) 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.

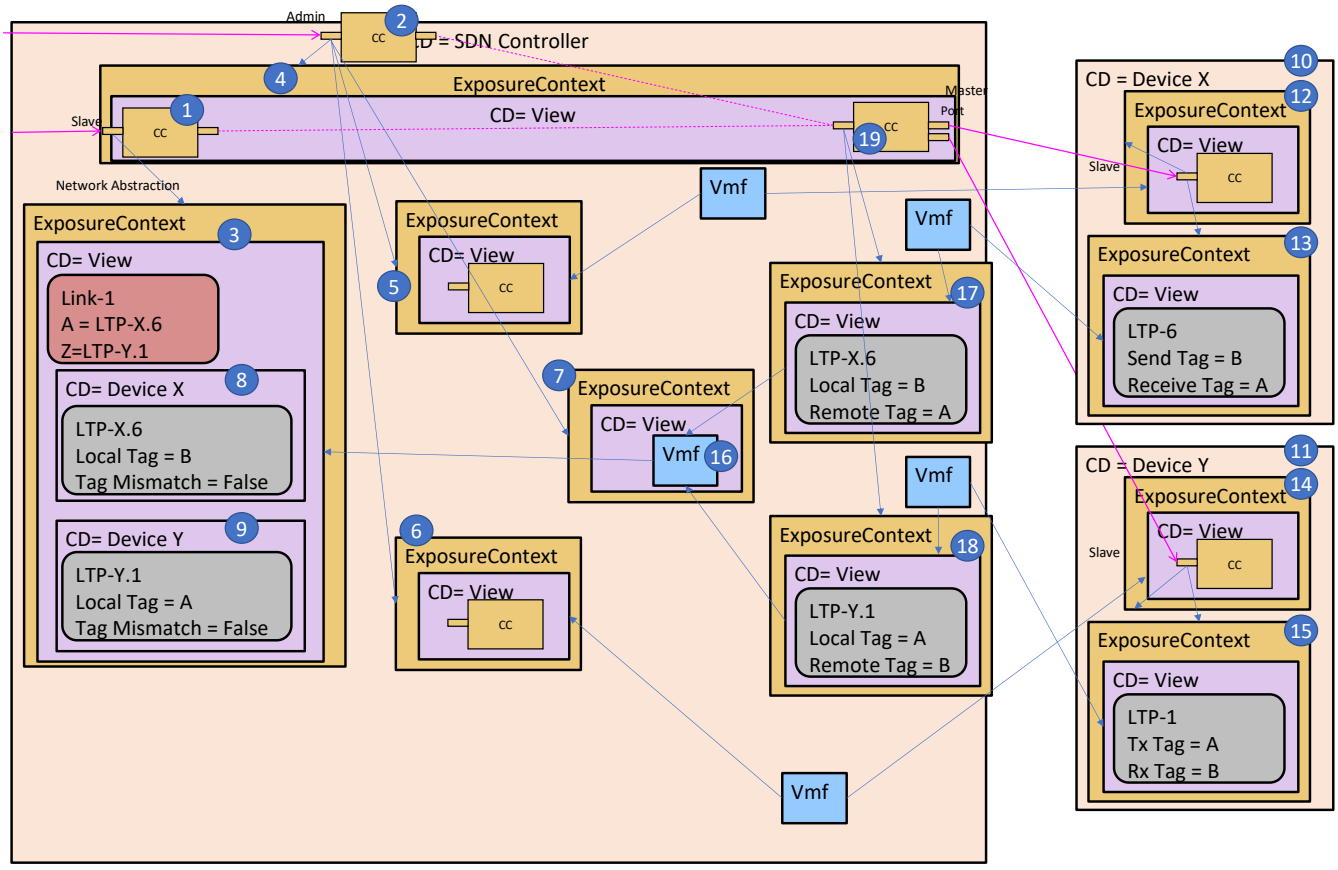


Figure 3-13 – SDN Controller controlling two devices

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 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 of the capabilities of the devices (10 and 11) it controls and mappings of some of its own capabilities (i.e., of the capabilities of the controller itself) 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 may use some other model[9].

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 35).

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≠17 and 9≠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.

[9] The CIM could be used at all levels of view of networking capabilities. Clearly legacy devices will use traditional representation forms such as TL1.

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 simply 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[10].

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.

[10] 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.

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.

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[11] (which were potentially simply control components with broader scope)
- Highlighted recursive functional abstractions, where a selection of functional components offered by providers[12] are taken by a client, pruned to give useful function, assembled

[11] The client of a control system is the entity that makes requests for action by the control system.

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. The 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. A control device and a transport NE have significant similarities, both have communications ports and control functions. In both cases, the communications capabilities are used to support control functions. The traditional Network/Managed Element has additional communications capabilities that are provided to a client. In a traditional Network/Managed Element some of the control functions act directly on the transport capabilities (e.g. to support APS, justification control etc.[13]). All devices are balances of compute, storage and communications capabilities (it is just the specific balance and use of those capabilities 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 aggregated within a related ConstraintDomain, and not via a 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.

---

[12] The provider of a control system is the entity that offers capability to the control system for control of and/or reporting about some resources.

[13] In general, a control function acts to modify the action of a component based on a set of input conditions, thus functions such as APS and justification control are control functions. Whilst the control model can be applied to any control function typically it is not used to represent “low-level” control functions that act directly on the transport capabilities. These low level control functions typically dedicated messaging channels embedded in the transport overhead.

If there is an opportunity to see the native model of the NE as well as the mapped model, then the ControlConstruct that represents the NetworkElement will also have an 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[14]:

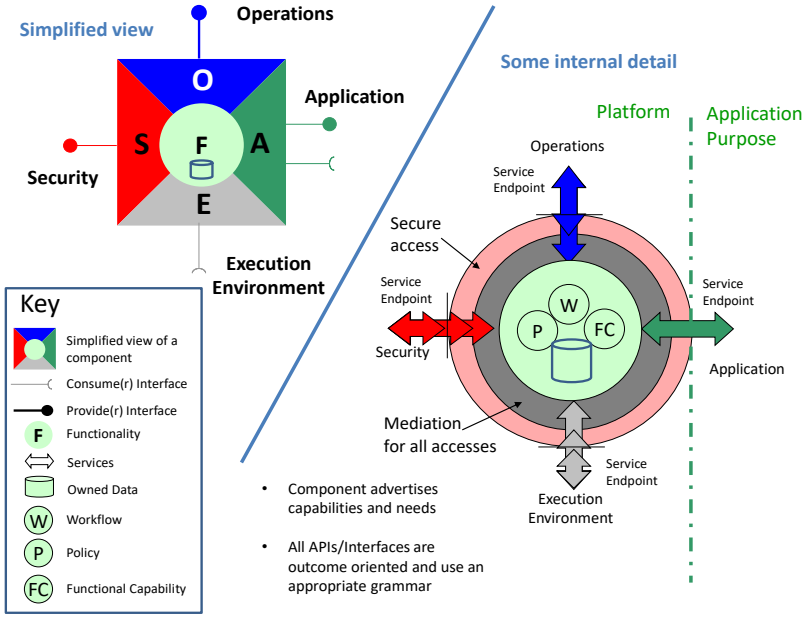
- A logical scope that aligns to a physical inventory boundary (especially useful for 'device partitions' and 'distributed devices')
- A management scope (which may differ from the physical and functional scope)
- A general functional scope that can be used for grouping and scope boundaries

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 traditional "physical" 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

[14] The constraint domain scopes the set of resources, the exposure context defines/limits the control capabilities that are offered to the user and/or provides an abstracted view and/or different name space.

A Component has an Operations port through which it may be controlled/managed[15] 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. CASC).

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

[15] 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).

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

A ControlPort has operations that are about the entities (aggregates) available through the ExposureContext, the entities do not have operations as the client does NOT talk to them. The client controller talks to the provider controller about the controlled things.

The control port supports the interface between the client and provider controllers. The control port supports provider and user roles. The signals and operations (confined to the control port) are about what is exposed. The control port is not modified and is not signaling about itself. It is providing information from the exposure context about the things represented in the associated constraint domain.

The control port is the place where interaction takes place. None of the other entities emanate any signals. It is the control port that signals about things (see section 0

Operations on page 68).

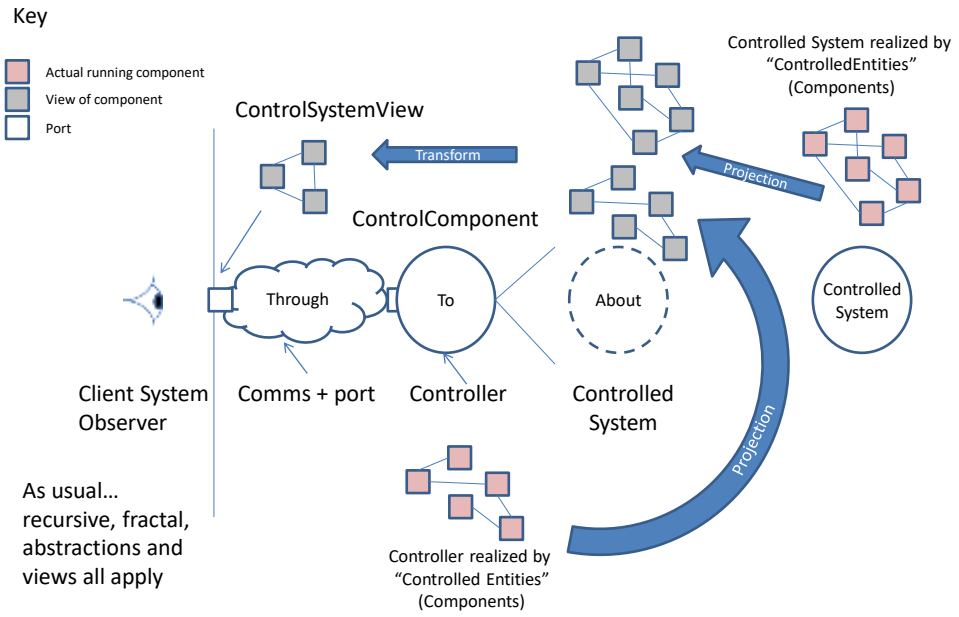


Figure 4-2 Through, To, About…

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 or how the function is supported/implemented in the network. The figure shows a network that has a function "B" that is exposed as "Func B' " to the Client.

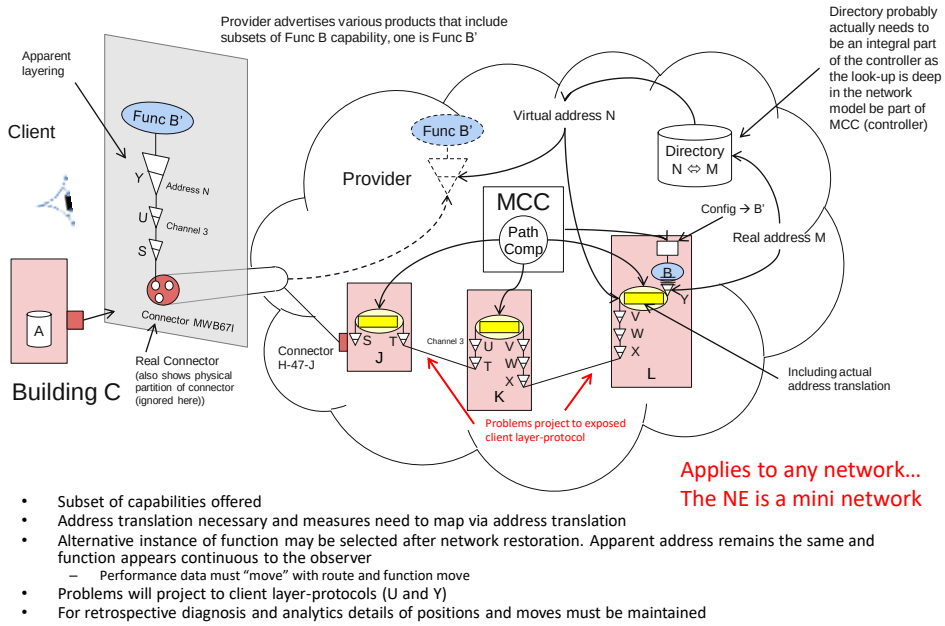


Figure 4-3 Simple network view mapping

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. Hence, the management of the network implementation is different, however, this difference will not be visible to the client.

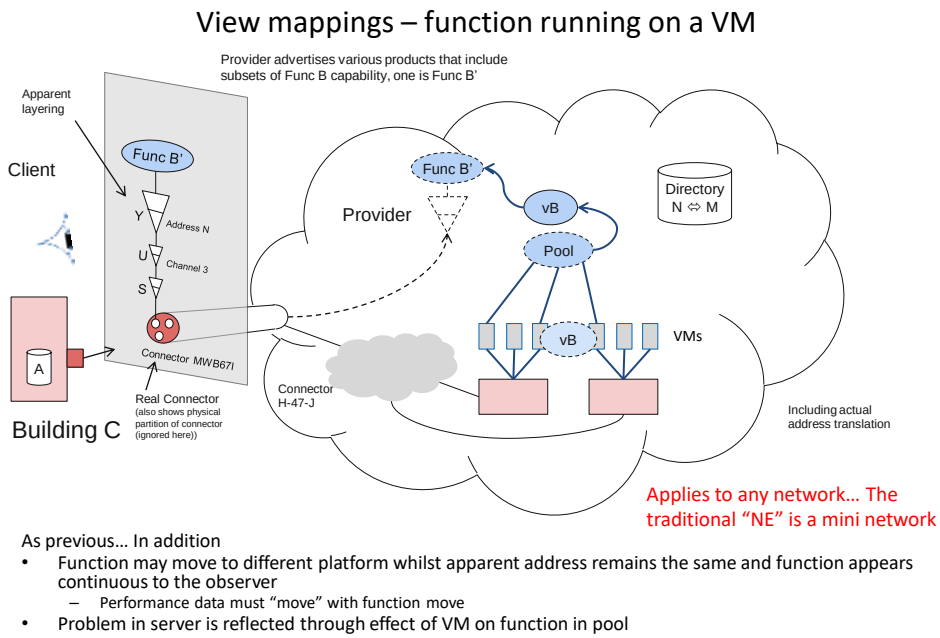


Figure 4-4 View mapping for functions on a VM

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.

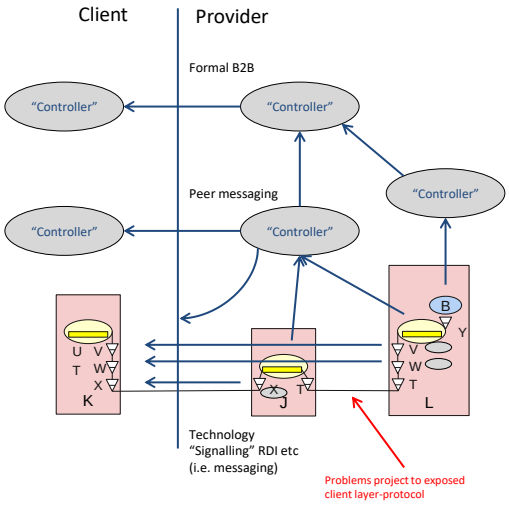


Figure 4-5 Client view of network and control

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[16] at the network technology level is "immediate" but provides minimal information and hence may cause 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]

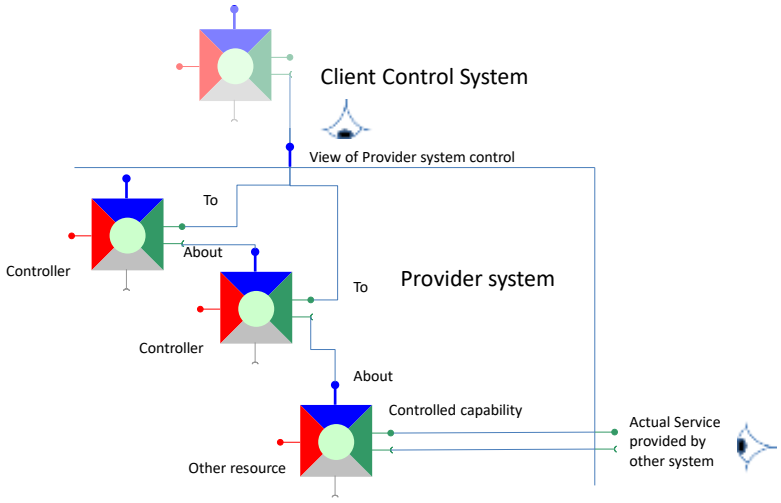


Figure 4-6 Simplified view showing exposure of controllable capability to a client

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[17] 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."

[16] Messaging could include APS signaling, justification control etc. In general, these “low level” messaging systems are, of necessity, immediate, and are quite robust. In these cases, the “immediate" action is the best option.
[17] https://definitions.uslegal.com/n/network-element/

- [ITU-T G.780] defines "network element (NE)" as "A stand-alone physical entity that supports at least network element functions (NEFs) …"

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.

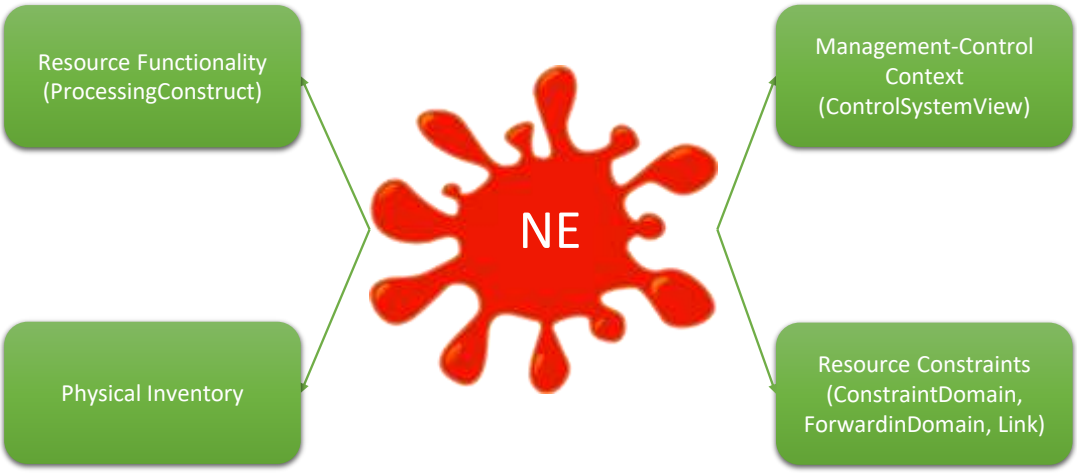


Figure 4-7 The "NE"

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 concept of the Network Element (NE) was created at the time when a single physical chassis (the NE) supported a single network function. Over time it became possible to support both multiple network functions in a single physical chassis and to support a single network function over multiple (geographically distributed) physical chassis. Thus, over time the Network Element (NE), as concept, became 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[18] function emergent from a physical thing (or set of physical things) 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)

[18] Also called Logical Function.

    - 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
- 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 ControlConstruct 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 ControlConstruct. 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.

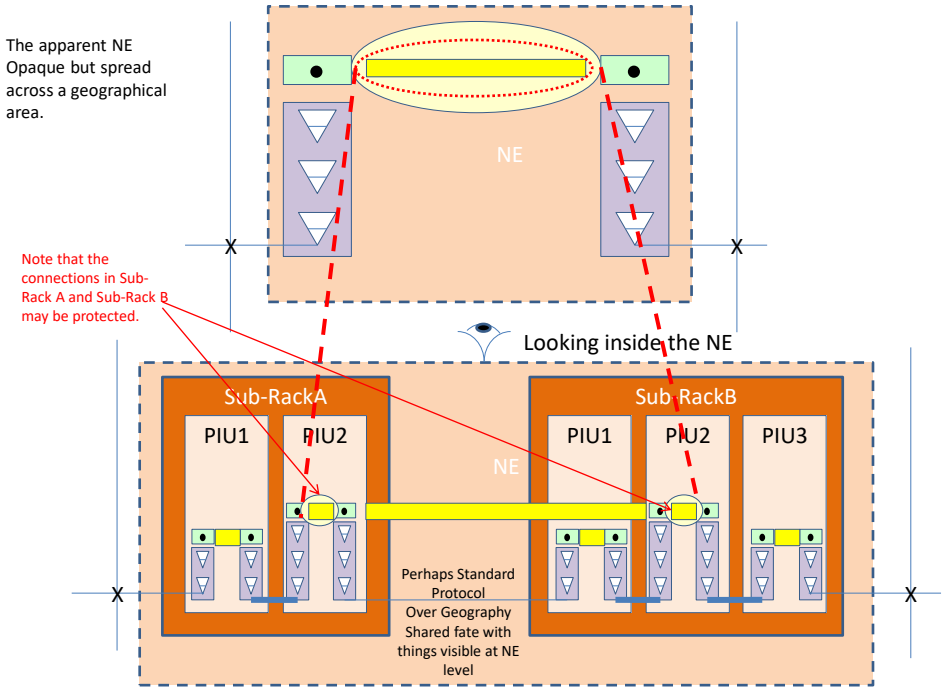


Figure 4-8 Geographically distributed NE

In the figure above:

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

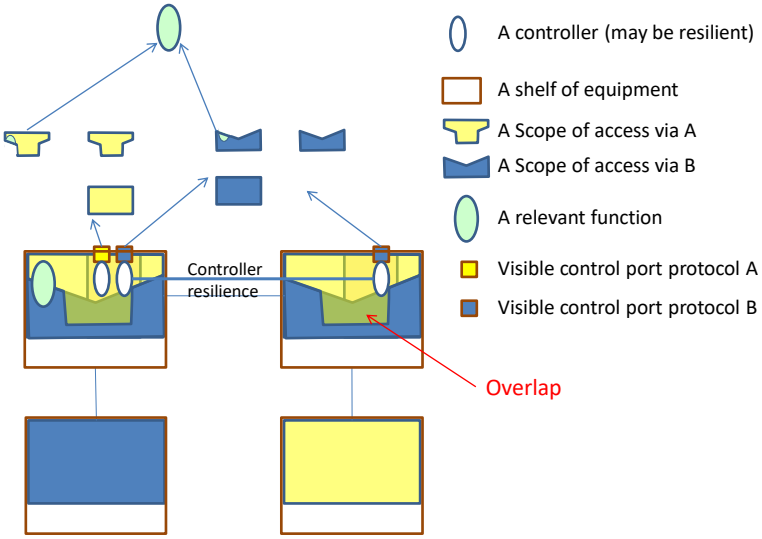


Figure 4-9 An NE with two control access ports each providing a partial view

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 and ConstraintDomain (see TR-512.11). The Controller model is not fully developed in this release.

4.9 The configurationAndSwitchController (CASC)

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

5 The ControlTask

5.1 Overview of Tasks

5.1.1 Task Definition

Task: A piece of work to be done or undertaken.

- Note that in the English definition, a Task can be any size or complexity (and is often considered as hard (not simple)).
- Clearly a Task can be broken down into smaller Tasks where there is therefore a (complex) arrangement (flow) of Tasks to perform a Task.

5.1.2 Examples of Task based infrastructure

There are many task-based infrastructures available in the marketplace but most appear to take a narrow view of Task. For example, Kestra (https://kestra.io/) appears to have a narrow usage in that it appears to assume a Task is a single action in a flow as opposed to a nesting of other Tasks. It does note that a "Runnable Task" can be compute intensive (and can be anything).

5.1.3 Examples of Task base solutions

The TAPI oam-job is essentially a Control Task related to measurement, where measurement is itself a Task.

5.1.4 Tasks in general

Tasks interrelate in some complex interconnected structure to perform control activities. A complex of Tasks may be abstracted to be viewed as a single Task.

A Task is defined in terms of dependent desired outcomes that themselves are defined in terms of constraints (as per intent). The outcomes may be explicit or implicit. The intention is that any task can be fully defined.

The control solution may choose to either expose tasks to any level of detail/abstraction. For simple operations there will be no relevant Task exposed (although there clearly is a task).

A Task can itself be controlled (via another Task), so an outcome of PAUSED may be requested. If this takes time, then a visible Task will administer the control of the Task (it is assumed that this Task that administers the pausing of another Task will not offer the opportunity for pausing)

A Task will operate over some limited time with a start and end, and potentially periodicity etc[19].

The ControlConstruct is the entity that carries out all tasks[20] in the control solution. The ControlConstruct carries out ControlTasks. The ControlTask provides an overarching entry point to the Task flow etc.

Control of the task flow is essentially Task orchestration and hence the overarching aspect of the Task.

[19]The end may not be known till just before it occurs, but nothing goes on forever.
[20] All functions, activities, operations etc., i.e., all behavior.

5.1.5 Task lifecycle

The Task lifecycle can be represented in terms of a set of states that relate to progress through the Task. As a complex Task may have many branches, parallel activities and alternatives. The lifecycle expression will have the complexity necessary to describe the task.

5.1.6 Context for a Task

A Task can act upon a subset of the resources made available through an ExposureContext from an associated ConstraintDomain. The Task operations (creation, deletion, adjustment etc.) relate to the capabilities of the resources made available. The Task will be defined in terms of that subset of resources and a set of other Tasks it can interact with.

5.1.7 Purpose of a Task

The Task purpose is defined in terms of a set of outcomes (may be instantaneous, ongoing (where a short-lived Task caused an ongoing Task) etc.). This outcome is defined in terms of other model entities and is constraint based.

5.1.8 Outcome of a Task

An outcome of a Task is defined in terms of a space of constraints. The constraints define a boundary within which the outcome must be. An outcome may be essentially constrained capabilities.

An outcome may be broken down into many individual and more specific outcomes. An outcome may be defined in terms of effects, things etc. The things may themselves be performing an ongoing activity, for example things creating other things etc. An FC, resulting from a request for forwarding, represents things that are performing the ongoing activity of forwarding. Forwarding is a Task (although NOT a Control Task).

The Task constraints bound the outcome and define the "intent" for the Task. TAPI connectivity-service is an example of Task outcome constraints.

An outcome maybe:

- The experience of something happening, e.g., the called party appears to the calling party to be in close proximity.
- Something happening
- The experience of something having happened
- Something having happened

A measurement job (resulting from some request for measurement), performs the ongoing activity of measuring. The measurement job is therefore also a Task.

Confirming and ensuring a successful outcome will often require measurement. Hence, many outcomes will involve the creation of Task related to measurement.

Note that the TAPI oam-job is essentially a Control Task related to measurement.

5.1.9 Expressing the Task

A Task is defined in terms of a structure of expression of capability and action (e.g., get, set etc.). That expression is contained in an envelope with nesting and interrelationship of opportunities. A grammar is necessary to enable the expression of things performing the Task and expression of the Outcome of the Task. The expression is in terms of constraints, relationships between things, temporality etc. and this expression will define all potential lifecycles for the Task. Task constraints can be expressed in terms of the OperationEnvelope model TR-512.10 (which provides some interdependencies related to the specific overarching intent). The description of the task is essentially a workflow description that is related to the instance of Task.

The OperationsPattern TR-512.10 has OperationSet and dependency which is essentially a dependency skeleton of the Task.

5.1.10 Task activity sequence

There is a general need beyond expression of a single Task that supports various degrees of Task coupling. At one end of the spectrum there is extremely loosely coupling where a Task results in an outcome that makes another Task more likely to be triggered. At the other end of the spectrum there is tight coupling where the outcome of one Task is guaranteed to trigger another specific Task. This appears to be a confluence of policy and workflow.

Note that further work will be required to develop a description of policy/workflow confluence (which may draw heavily from external work).

5.1.11 Task results as a specific type of outcome

A Task may have results, for example, in the case of a Task related to testing there may be some test results. These test results should be modelled, should be related to the Task but should be separate from the Task. The results should be maintained in a Task history in the context of a broader general history for them to be meaningful.

The Task results may be in the form of the functions that perform other ongoing tasks.

A measurement Task may require an ongoing record of measurement context (and hence a chronological log of change) which itself requires a Task.

Note that further work will be required to develop a model of measurement results and of a historic log.

5.1.12 Task as a Component

A Task is a definition of functionality and hence can be viewed as a Component as described in TR-512.A.2. A Task:

- has inputs and outputs
- can be adjusted with policy and controls
    - In the case of the control task, these are all externally visible and provided via inputs.

- has internal workflow
- is described in terms of subordinate components
- etc.

Hence a Task instance can be considered as an active functional component interacting with other components.

5.1.13 Task structure in more detail

A Task may breakdown into subordinate Tasks where those Tasks may be serial, parallel, have interdependent starts etc. (as per [BPMN]).

A Task may cause several outcomes where some outcomes may occur at some intermediate point in the Task run and where some outcomes are a change of state that trigger or influence other Tasks or the Task itself.

The Task is an abstraction of the underlying function arrangement (the underlying processes). Any functional entity can be considered as a Task Function

- The ProcessingConstruct is useful for general Task Functions that are not being modelled fully
- The ForwardingConstruct carries out the ongoing Task of transfer of information
- The ControlConstruct carries out an ongoing control task
    - There is a need for further control function representation
    - Could use decorated PC or specific new classes

Running these functions may require monitoring where monitoring is itself a Task

The statements of desired outcome may include interdependent activities to be completed (where the interdependency may be sequencing or finish before start etc.).

5.1.14 A flow of Tasks

Flow in one Task is initiated by trigger conditions (being watched out for by that Task) where those conditions may be caused by outputs from other Tasks or from other sources.

If multiple Tasks are watching for the same trigger condition, then the tasks run in parallel in multiple branches after the trigger. The flow may rejoin if a Task is watching for and requires two or more specific condition outputs (one from each of the two or more branches) to satisfy the trigger condition.

A Task may take a set of inputs, process them, provide a set of outputs (or achieve outcomes) then complete/terminate.

- The outputs may all occur at the completion of the task, or some may occur at intermediate points
    - The outputs may directly update system state or may be streamed for use by other components

- The inputs may all be available at the start of the Task, or they may be available at various points
    - The task will be initiated by the occurrence of some condition (trigger)
    - The inputs may be from monitored state or monitored stream
    - The task may pause to wait for an input, abandon if it does not have an input, skip the input etc.

A Task instance may run as a single activity that terminates once complete or, run continuously with internal loops until requested to terminate via some state input depending upon the Task capability.

5.1.15 Task capability

The Task capability may be expressed in terms of apparent task flows that explain, in abstract, how the outputs are generated from the inputs, i.e., the definition of the transfer functions.

The transfer function may be expressed as a structure of apparent encapsulated Tasks with some stated flow. As for any structure of tasks there may be flow loops and the apparent Tasks may themselves have capability expressed as transfer functions. The transfer function may be expressed as some other structure that is not of a task.

The Task may be realized by subordinate Task flows built from a structure of real tasks with stated flow. Flow is determined by trigger conditions that are caused by outputs from other tasks etc. as discussed in section 5.1.14 A flow of Tasks on page 53.

The Task may be realized by code (algorithms etc.) in which case there will be no deeper view of realization. In this case there still may be an expression of capability in terms of apparent encapsulated tasks with some stated flow.

5.1.16 A running Task

Multiple instances of a specific type of Task may run concurrently. A Task instance is run in a specific instance of flow and will be related to other instances in the same instance of flow. Clearly, there is a need for an identifier for a flow where that identifier system deals with the nesting of flows etc. A running Task can watch for a trigger event to cause it to take a step etc. Instantiation of a running task is triggered by an event and hence there must be some form of task manager to instantiate the Task. This Task manager is essentially a running Task that has the role of Task instantiation. Clearly the Task manager Task needs to be started (see section 5.1.19 System Initialization on page 60). This area requires further development.

The figure below shows tightly coupled Tasks where one Task triggers the next (i.e., where a Task is waiting on a condition to be triggered and where that condition is explicitly set as an outcome of another Task). The Tasks could be ControlTasks.

Note that the Task flow statement is considered as a Task, albeit a simple one.

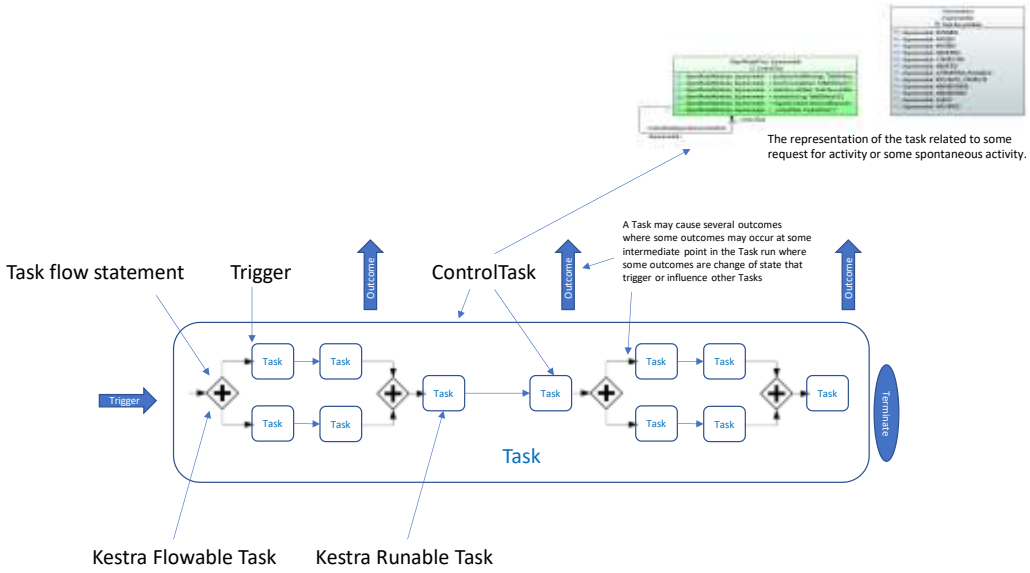


Figure 5-1 Tasks and triggers illustration

The figure below shows an example of a Task flow.

The trigger is a new intent request. The request will carry the constraints defining the desired outcome. The task is shown broken down into apparent subordinate tasks (that may happen to closely reflect the actual underlying task realization).

The trigger causes an instance "Evaluate Request Integrity" to run thus initiating some instance context (identifier of the specific instance of the task), e.g., "Intent_x".

As a result of integrity having been validated (the failure path is not shown), a "process request" event occurs and this triggers both the "Evaluate Realization Template" and "Gather Policies" (as both are waiting on the same event. Both run an instance for the specific instance context, "Intent_x".

At some point in the execution of "Evaluate Realization Template", the event "realization policies" is caused to occur in the context "Intent_x". This is absorbed by the "Gather Policies" for "Intent_x".

The flow proceeds to eventually reach the "Evaluate Computation Outputs". This is initially triggered for "Intent_x" by two triggers, "compute A" and "compute B" causing it to start then wait for both the "Compute A Result" and the "Compute B Result" triggers. If both do not arrive appropriately the "Evaluate Computation Output" escalates.

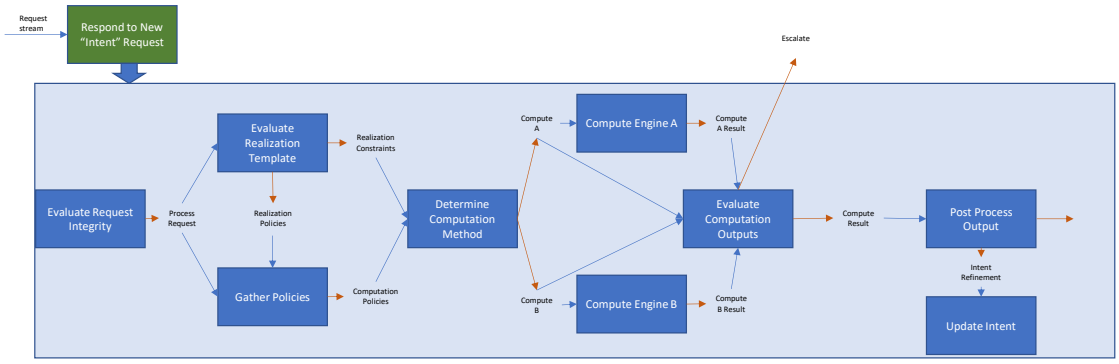


Figure 5-2 An example ControlTask flow

5.1.17 The ControlTask

The ControlTask in the Core Model is intended to support activities prompted by client interaction with the ControlConstruct. The ControlTask instance provides an opportunity for the client to assess progress and to adjust the ControlTask operation. Modification of the direction for the activity should be carried out via the ControlTask.

Clearly, the client observes and manipulates a shallow abstraction of the actual Task complexity of the provider solution. The Tasks that appear to act on the entities exposed via the ExposureContext are mapped from the actual detailed Tasks.

Different clients may get different degrees of visibility. An external party may get a very lightweight view (e.g., on/off) whereas an administrator of the actual solution will see the detailed Tasks at one level and potentially the mapping between the levels, depending upon the administrator role. Clearly, the solution designer will have an even more detailed view of the Tasks.

ControlTask capability is defined from the outside and hence its description does not vary due to internal hidden control. Other components expose capability that is defined from the inside.

5.1.18 Example of some high level ControlTasks

The following diagram, inspired by earlier work in TM Forum, shows a stylized sketch of control Tasks that are arranged to perform service design and provisioning[21]. These Tasks will not all be visible to the administrator. The client is likely to see a thin shim of the Task behavior where some of the stages are identified, perhaps that placement computation is complete, but probably not that the Solution Interpreter is active.

[21] It is not intended that this be a full/complete design, the structure is simply being used to illustrate the mechanism.

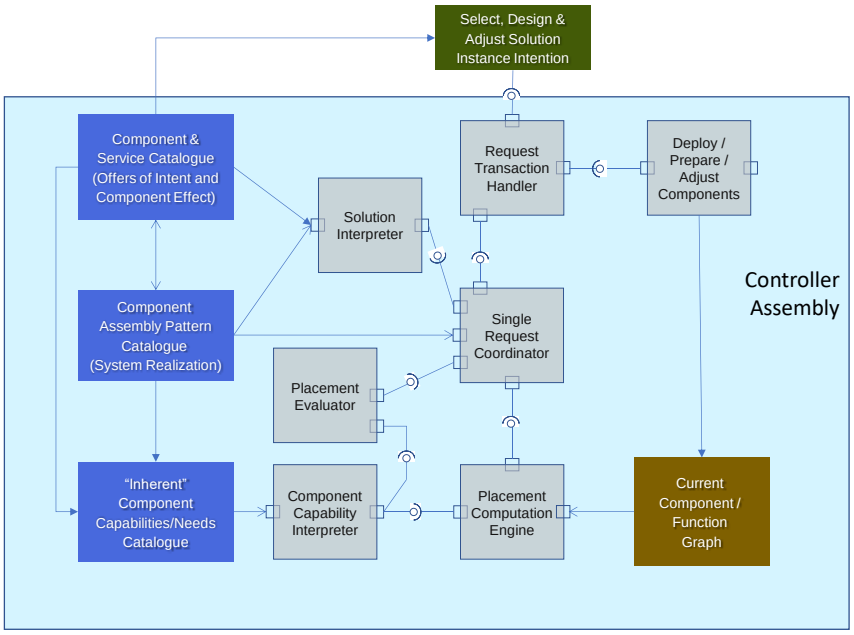


Figure 5-3 An example ControlTask structure in a Placement solution

The sketch shows three repositories (blue) holding catalogues of specification data that define types related to Service Offers, System Realizations and Component Capabilities. There is also a repository (brown) holding a cached view of the controlled solution in terms of a live graph of component/function instances which show current resource usage in the controlled solution. This can be considered as a network inventory. Information in each of these repositories is represented using Core model entities. The combination of the four repositories provide the information necessary to drive a network digital twin (where that would also have a view of planned and candidate futures instance structures etc.).

The grey components are ControlTasks. The sketch does not show the ControlConstructs, ExposureContexts etc., but instead focusses on the ControlTask interaction.

The following diagram shows the control flow through the tasks.

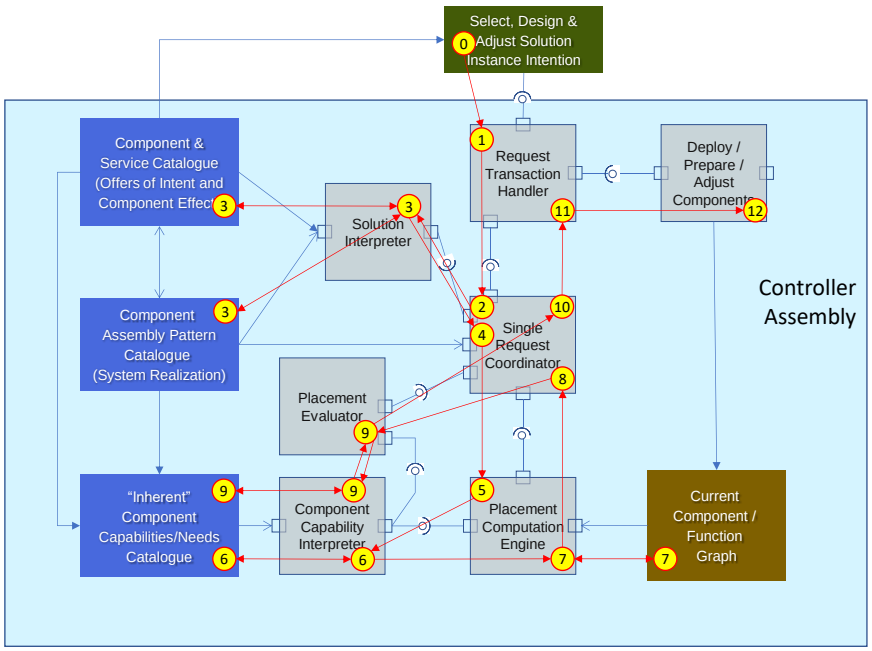


Figure 5-4 An example ControlTask flow in a Placement solution

At the top of the sketch, an external initiating selection function makes a request on a controller for one or more specific services. This would arrive at a ControlPort. A combination of ControlConstructs run the following ControlTask flow (note that this is clearly a massively simplified sketch of an essential flow):

1. Request Transaction Handler: Breaks multiple requests into a set of single requests
2. Single Request Coordinator: Takes a request and asks for interpretation against known offer types (Component/Service) and candidate realizations.
3. Solution Interpreter: Provides one or more candidate solution design patterns for the requested Component/Service
4. Single Request Coordinator: Provides the Placement Computation Engine with the candidates
5. Placement Computation Engine: Uses Component Capability Interpreter to evaluate capabilities and needs related to the solution design patterns
6. Component Capability Interpreter assesses each request against the definitions of Components in the catalogue
7. Placement Computation Engine: Interrogates the Current Graph for opportunities and runs some algorithm etc. to determine appropriate placement of necessary components etc. using the Component Capability Interpreter as appropriate and returns candidate solutions to the Single Request Coordinator
8. Single Request Coordinator: Takes candidates and requests evaluation for suitability.
9. Placement Evaluator: Assesses quality of each candidate using the Component Capability Interpreter as appropriate and provides assessment back to Single Request Coordinator

10. Single Request Coordinator: Chooses solution and provides this to Request Transaction Handler
11. Request Transaction Handler: Initiates deployment etc. of the solution as appropriate
12. Deploy / Prepare / Adjust: Takes necessary action including updating the Current Graph as necessary

The above flow may be augmented with Policy considerations, schedules, lead time considerations, negotiation phases related to third party network requests etc. It is clearly part of multiple control loops that relate to reevaluation on changes in the solution environment, changes in request etc.

The diagram does not show failure paths and refinement loops (e.g., where the Single Request Coordinator determines, using the Placement Evaluator that none of the placements offered by the Placement Computation Engine are suitable so the Single Request Coordinator requests further placement attempts by the Placement Evaluator, perhaps with refined properties in the request.

The diagram does not show multiple parallel requests and does not cover what happens to other members of the set of single requests that the Request Transaction Handler constructs. It is assumed that occurrences of each necessary task will run for each of the single requests and that these may run in parallel. Clearly, there may be some dependency between results, e.g., the case where if one of a set of computationally independent single requests cannot be successfully resolved, then the whole set should be abandoned.

It could also be argued that step (1) and step (11) are run by separate tasks, i.e., coordination is distributed and information on context etc. flows forward with the progression of the other tasks (as opposed to flow back to the same originating task). Clearly, step (11) needs to be run by a stateful task that is aware of progress and of the definition of completeness.

As discussed in section 5.1.16 A running Task on page 54, the coupling between tasks could use an event strategy such that a task handler is listening for a particular trigger event that will then cause the spawning of a new task occurrence and each live task occurrence could be listening for events to enable progression (and of course, termination).

A full solution will also support control behavior, not shown in the sketch above, related to:

- network failure, where a recomputation of placement of a running service/component is performed to recover the support for the intended solution characteristics
- changes in underling network infrastructure supporting a running service/component, where a recomputation of placement of a running service/component is performed to recover the support for the intended solution characteristics
- concurrent user computation result clash, where two users both attempt to acquire the same resource
- deployment failure, where there is some conflicting deployment in the network or some other real network issue
- modification of an existing intent resulting from negotiation with the client
- modification of the component assembly pattern supporting a particular offered component/service type

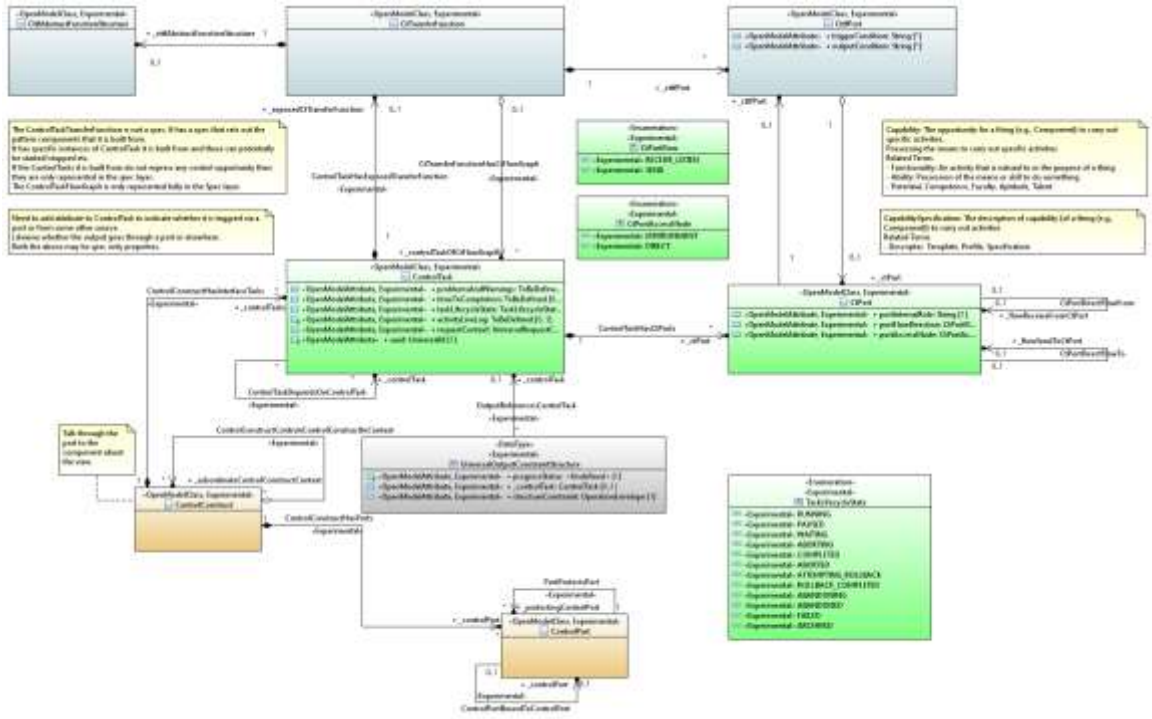
- modification of capabilities advertised for a particular component and hence made available to the Placement Computation Engine and to the Placement Evaluator

5.1.19 System Initialization

As implied earlier, there needs to be a ControlTask running at startup to instantiate other ControlTasks etc. It is assumed that at system initialization there will be a ControlConstruct running with a specific ControlTask that will boot the system. This ControlTask will have the purpose of creating ControlTasks that will create the initial CDs, ECs etc. The initial CDs/ECs/etc. will probably be related to the system control functions and each will cause the formation of further CDs, ECs etc. After several sequences of CD/EC/etc. creation, the CD/EC/etc. related to the core purpose of the system will be created.

The specific arrangement and sequence of assembly will depend upon the purpose of the system.

5.2 The ControlTask model



CoreModel diagram: Control-ControlConstructWithTask

Figure 5-5 Control Task Model

5.2.1 ControlTask

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::ControlTask

The ControlTask represents an apparent (abstract/emergent) functional Component that provides/exposes management-control capability where that capability is defined in terms of a CtTransferFunction (Control Task Transfer Function).

The use of "apparent" and "emergent" emphasizes that this is NOT the underlying/implementation componentary.
- The behaviour defined by the CtTransferFunction is emergent from a set of underlying implementation components that have been constrained to behave in a particular way.
- The ControlTask defines a specific purposeful CtTransferFunction where the underlying componentry may be far more capable/complex. It has architected behavior.
- The ControlTask capability is defined from the outside and hence its description does not vary due to internal hidden control (other components expose capability that is defined from the inside).

The whole defined CtTransferFunction is available and active in an instance.

It achieves outcomes/goals etc. and covers all success and failure behaviors.

It is the representation of the behavior related to some request for control activity or some spontaneous control activity.

ControlTask capability definition and instance of running ControlTask with state

Related terms:
- Task: A piece of work to be done
- Job: A task or piece of work
- Activity: A thing that is done
- Use Case: A written description of how a task will be performed for a particular purpose
- Function: An activity that is natural to or the purpose of a thing
- Action: A thing done
- Runnable Task: Used to handle any computational work

Inherits properties from:
- GlobalClass

This class is Experimental.

Table 8: Attributes for ControlTask

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
problemsAndWarnings	Experimental	A list of problems and warnings related to the task.
timeToCompletion	Experimental	The estimated time to completion of the task.
taskLifecycleState	Experimental	The state of the task (progress etc.).
activityLiveLog	Experimental	A log of activities.
requestContext	Experimental	All details from the request.
_controlTask	Experimental	See referenced class
_exposedCtTransferFunction	Experimental	The exposed behaviour of the ControlTask.
_ctPort	Experimental	Provides a specific access. ControlTasks can only be connected by ports. A ControlTask may have no ports when it is not explicitly in some ControlTaskFlowGraph, i.e., where there is no explicitly stated trigger and no explicitly stated output to cause flow progression.

5.2.2 CtPort

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::CtPort

An access to a ControlTask.
This class is Experimental.

Table 9: Attributes for CtPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
portInternalRole	Experimental	The aspect of the ControlTask functionality accessed via the port.
portFlowDirection	Experimental	The direction of flow (in or out).
portAccessMode	Experimental	The mode of access to the port.
_cttfPort		See referenced class
_flowSendToCtPort	Experimental	This SEND control port is explicitly DIRECT connected to the referenced (RECEIVE_LISTEN) ControlPort. There may be a DIRECT connection but it may not be known at the SEND end (hence this attribute may be not present).
_flowReceiveFromCtPort	Experimental	This RECEIVE_LISTEN control port is explicitly DIRECT connected to the referenced SEND ControlPort. There may be a DIRECT connection but it may not be known at the RECEIVE_LISTEN end (hence this attribute may be not present).

5.2.3 CtTransferFunction

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::CtTransferFunction

A statement of the capability of the ControlTask in necessary detail to enable a client to fully understand the externally visible characteristics of the ControlTask (i.e., how the outputs are generated from the inputs, or from any other relevant internal behavior).
Each apparent ControlTask should have a defined CtTransferFunction.

It may be expressed in terms of:
- an apparent Flow Graph that explains, in abstract, how the outputs are generated from the inputs.
- logic function, arithmetic function or some other structure (CttfAbstractFunctionStructure) that is not in a ControlTaskFlowGraph form.

The _controlTaskOfCtFlowGraph collects the ControlTasks of the Flow Graph.

Related Terms
- Behaviour: The way in which a thing works
- Transfer Function: The relationship between the output signal of a control system and the input signal

The CtFlowGraph (not modelled directly) is a structure of interconnected apparent/abstract

ControlTasks (each having a defined CtTransferFunction) where the structure expresses all possible flows (including cycles/loops) from exposed inputs to exposed outputs (which are the inputs and outputs of the ControlTask the ControlTaskFlow defines).
The CtFlowGraph is formed from the linking of CtPorts using _flowReceiveFromCtPort and _flowSendToCtPort (of the CtPort of the ControlTask).
The CtFlowGraph supports the workflow of the Component.

Related Terms
- Workflow: the order of the stages in a particular work process
- Flow (Kestra): A simple list of tasks for Kestra, grouped by namespace. It defines all the behavior for the current flow.
- Use Case sequence: a list of actions or event steps
- Process: a series of actions or steps taken in order to achieve a particular end
- Procedure: a series of actions conducted in a certain order or manner
- Action Steps: Detail of an action plan

This class is Experimental.

Table 10: Attributes for CtTransferFunction

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_cttfPort		See referenced class
_controlTaskOfCtFlowGraph		See referenced class
_cttfAbstractFunctionStructure		See referenced class

5.2.4 CttfPort

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::CttfPort

Represents the transfer function aspects of the CtPort
This class is Experimental.

Table 11: Attributes for CttfPort

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
triggerCondition		The condition that triggers the ControlTask through the related CtPort.
outputCondition		The condition output by the related CtPort.
_ctPort		See referenced class

5.2.5 CttfAbstractFunctionStructure

Qualified Name:
CoreModel::GeneralControllerModel::ControlTask::CttfAbstractFunctionStructure

Expression of a ControlTaskTransferFunction in terms of a logic function, arithmetic function or some other structure that is not in a ControlTaskFlowGraph form.
This class is Experimental.

5.2.6 CtPortFlow

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::CtPortFlow

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- RECEIVE_LISTEN:
  - An input port to the ControlTask.
  - Applied stereotypes:
    - Experimental
- SEND:
  - An output from the ControlTask
  - Applied stereotypes:
    - Experimental

5.2.7 CtPortAccessMode

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::CtPortAccessMode

The form of access.

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- ENVIRONMENT:
  - The port accesses the general environment. There is no explicit connection.
    A RECEIVE_LISTEN port takes input from anywhere and examines for appropriateness. There is no expected source.
    A SEND port broadcasts into the environment. There is no expected recipient.
    Analogue: An antenna.
  - Applied stereotypes:
    - Experimental
- DIRECT:

- There is a connection.
  The connected port may be known and/or may know of this port.
- Applied stereotypes:
  - Experimental

5.2.8 TaskLifecycleState [22]

Qualified Name: CoreModel::GeneralControllerModel::ControlTask::TaskLifecycleState

The potential states of the task.

Applied stereotypes:
- Experimental

Contains Enumeration Literals:
- RUNNING:
  - The task is running/progressing.
  - Applied stereotypes:
    - Experimental
- PAUSED:
  - The task has been paused.
  - Applied stereotypes:
    - Experimental
- WAITING:
  - The task is waiting for input etc.
  - Applied stereotypes:
    - Experimental
- ABORTING:
  - The task is aborting.
  - Applied stereotypes:
    - Experimental
- COMPLETED:
  - The task has been completed successfully.
    There may have been warnings and non catastrophic error conditions none of which prevented completion or some useful outcome.
  - Applied stereotypes:
    - Experimental
- ABORTED:
  - The task has been aborted.
  - Applied stereotypes:
    - Experimental
- ATTEMPTING_ROLLBACK:
  - The task is attempting to return the controlled system to a previous state.

[22] A task instance is run to completion. Once COMPLETE the task instance can be deleted or archived but cannot transition to any other state.

    - Applied stereotypes:
        - Experimental
- ROLLBACK_COMPLETED:
    - The task has completed a roll back action.
    - Applied stereotypes:
        - Experimental
- ABANDONING:
    - Task is abandoning.
    - Applied stereotypes:
        - Experimental
- ABANDONED:
    - The task has been abandoned and is no longer running.
    - Applied stereotypes:
        - Experimental
- FAILED:
    - The task has failed.
      The task may have failed to start or may have progressed to a point where a condition occurred that prevented further meaningful progress and has had no useful outcome.
    - Applied stereotypes:
        - Experimental
- ARCHIVED:
    - The task has been archived (and is no longer running).
    - Applied stereotypes:
        - Experimental

6 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.

6.1 The basic model

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

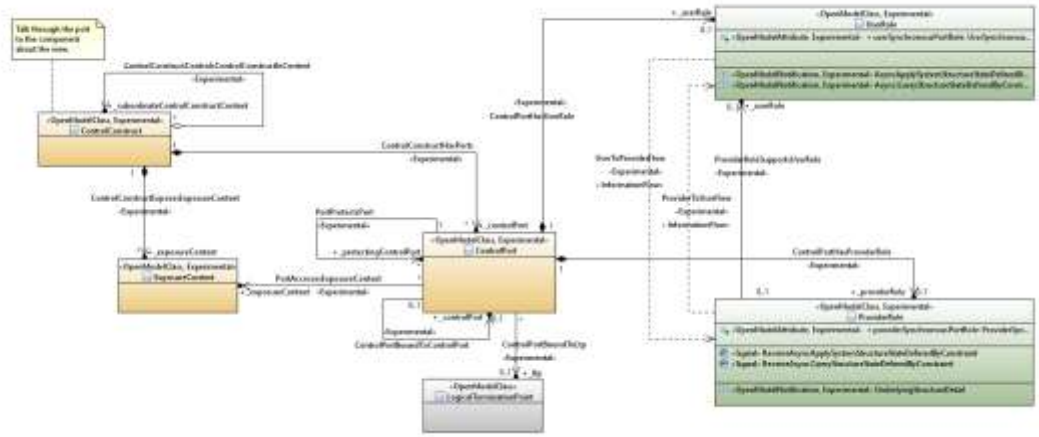
The ProviderRole offers:

- A synchronous interaction opportunity for traditional message/response interaction covering both:
  - Request for information
  - Request for change to be made[23]
- 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
- 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 6-1 Provider and User role ControlPorts

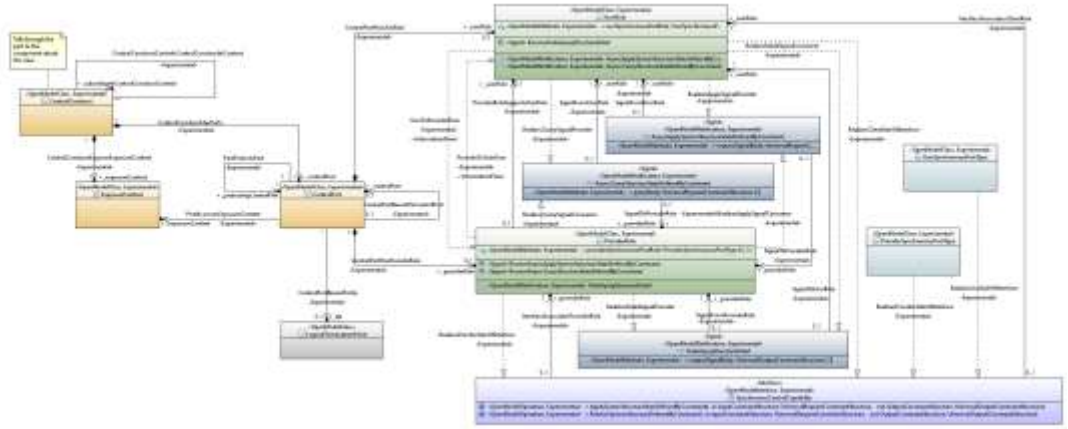
[23] Change may change a value, cause linear behaviour, cause some rate of change etc.

6.2 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 6-2 Provider and User role detail

6.3 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.

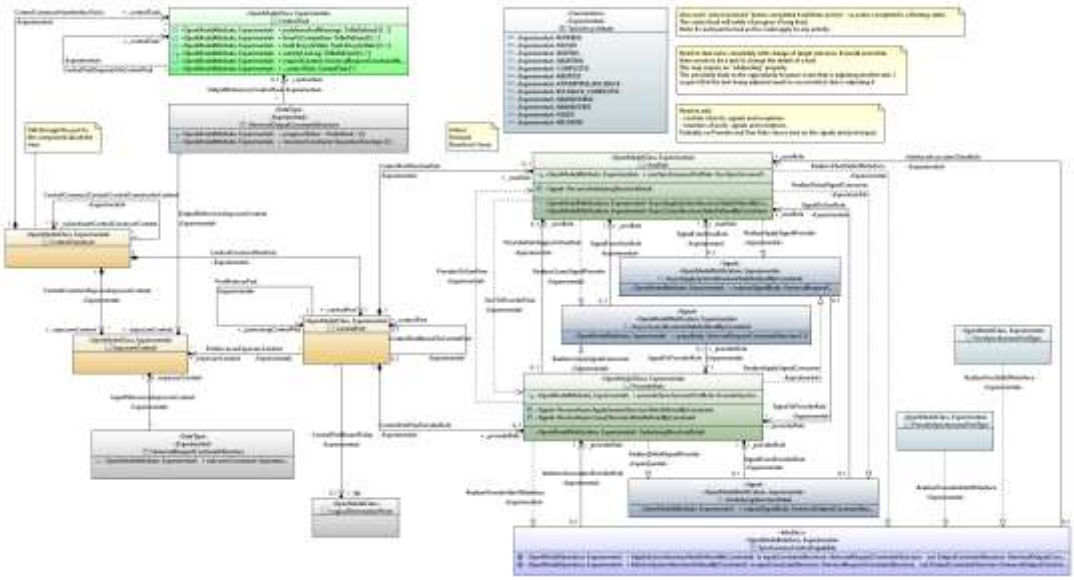
In cases where the task duration is long or is widely variable, initiation via a synchronous interaction is still possible but it is then necessary to provide a rapid response followed by updates on progress up to completion. The following figure introduces the ControlTask which will provide visibility 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 subgraph assemblies)
- ControlTask state and property changes where the ControlTask can progress through the any relevant TaskLifecycleStates.



CoreModel diagram: Control-ControlConstructWithUniversalStructures

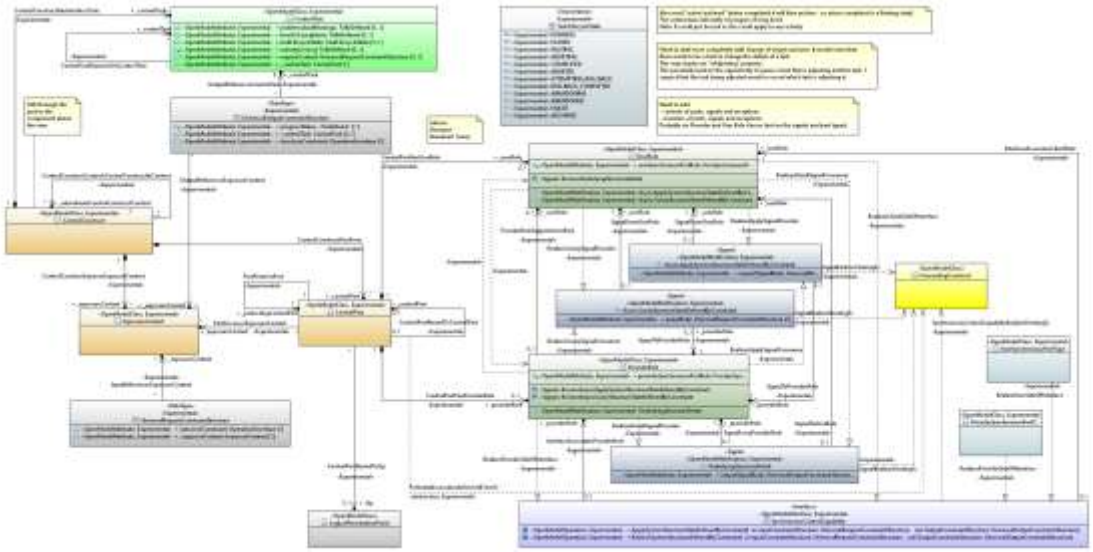
Figure 6-3 Long lived operations and universal structures

6.4 The full model

The ControlPort is supported by an LTP which will encode and propagate the relevant messages. Communication will be supported by the necessary connectivity represented by FCs associated to the LTP.

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 6-4 Full Control Model

7 Providing information

This section sets out the aspect of the control model associated with providing information through the signal "underlyingStructureDetail" discussed in the previous section.

This section focusses mainly on the autonomous provision of information but also considers the response to a request for information.

7.1 At the core of the Management-Control system

The following figure provides an overview of the controller structure and shows a Client Controller connected to a Provider Controller.

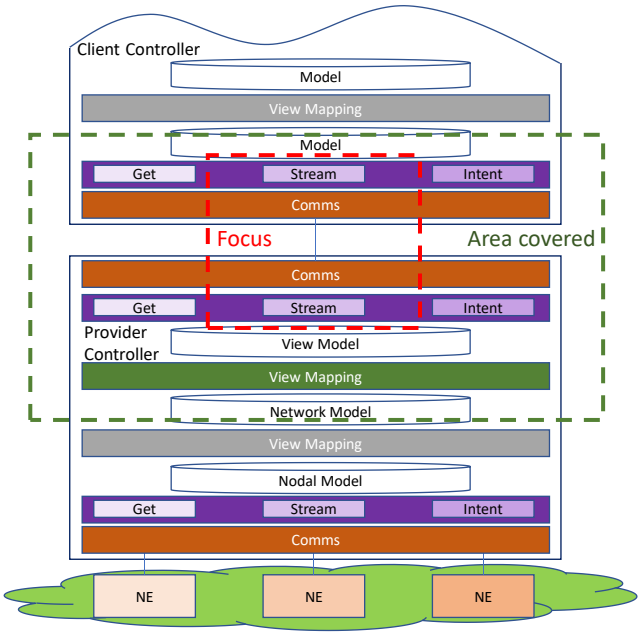


Figure 7-1 A controller hierarchy

A Management-Control system, such as an Orchestrator, high level controller, OSS etc., has the role of configuring and adjusting the controlled system (network) to achieve intended capability (intent, service etc.). By monitoring and processing information (e.g., alarms) from the controlled system, the overall assembly of Management-Control systems can determine actions necessary to enable ongoing support of intent/service. The Management-Control systems can also identify repair action prioritization (via analysis of problems).

Management-Control system components (in the client controller) interacts with components in a provider controller to acquire, from the provider, information from a fragment of the overall network, e.g., the devices controlled by a controller, where that information is presented in terms of modeled aggregates within an exposure context.

The exposure context will provide a view of the underlying controlled solution sufficient for the client controller to perform its necessary function. This view may be an abstraction of a subset of the underlying resources to reduce the burden on the controller. Presentation of a view for the purpose of control is covered in TR-512.A.15.

The client maintains history and live views of the state of the things in the network so as to do the necessary analysis, hence that Management-Control system uses a mechanism providing autonomous updates and need NOT query the provider for states.

There are several solution architectures that need to be considered:

- For interfaces low in the management-control hierarchy there will be many direct providers
    - For example, a controller may control many network devices where each offers some combination of fault, provisioning, equipment inventory etc. and potentially some form of resilience
- For interfaces in the middle of the management-control hierarchy there will be few (~2) direct clients[24]
    - For example, a single OSS/orchestrator with several separate internal systems (fault, provisioning, equipment inventory) and potentially some form of resilience
- For interfaces higher in the management-control hierarchy there may be many direct clients:
    - Interface to various business systems, for example a customer management solution, where each requires a special dedicated view
        - Each solution requires information of a distinct and different characteristic to each other
    - Interfaces direct to end user for viewing the resources that have been allocated to them (and potentially for configuration of those resources), where:
        - The end user will have a very small subset of the resources of the solution
        - The end user system may require the information to be partitioned across a set of specific focusses
        - There may be many end users
    - It is allowed for a provider to divide up the information based upon aggregate type
        - This allows simple separation of topology from equipment from alarms
        - This simple split probably matches the normal gross partition of roles in an OSS/Orchestrator
    - It is assumed that there will be one or two clients for each stream type (perhaps up to 4 if there is both an orchestrator and an OSS which are both providing some alarm capability)

[24] "Few (~2) direct client", is intended as order of magnitude, i.e. are not expected several tens of clients. In a future version there will be a broader consideration regarding client multiplicity for other applications

Considering solution integrity and recognizing that a controller has the role of maintaining the solution realization such that it satisfies the original agreement and hence the expectation of the end user, then:

- The Client that is closing the loop by making and acting on control decision has a long-lived connection with the provider of information:
    - Clients remain "connected" for a very long time and if the connection is dropped the same client will usually try to reconnect (e.g., by reestablishing a communication session).
    - Because of the point of use of the interface in the Management-Control hierarchy and the role and purpose of the clients (OSS/Orchestrator), it is expected that the clients will be permanently connected.
- The provider maintains alignment with underlying system
    - The realization assumes a reliable input that ensures eventual consistency (see section 7.7.8 Eventual Consistency on page 103) with current network state
    - As the client is an OSS/Orchestrator, it will have a repository.
        - The normal mode or operation is to align the repository with the view provided by the underlying system and to build a broader view of the network by integrating the views from many underlying systems.
        - For the OSS/Orchestrator to perform its expected functions it is necessary for it to maintain alignment.

In addition, for any of the specific considerations above

- It is assumed that a controller may be composed of functionally focused components (e.g., fault analysis, path computation) and that it may partition information across streams from these focusses but that it will provide a unified view to the systems to which it streams
- The controller may be operating some form of resilience and hence there may be several separate feeds for any particular information

The primary focus for Streaming is simple and efficient ongoing alignment of a client with a view presented by a provider.

7.2 Autonomous provision of information

The model covers a streaming approach. Streaming is the name for a type of mechanism that handles the providing of information from one system to another in some form of steady and continuous flow[25].

In the context of a Management-Control solution streaming is used primarily for the reporting of ongoing change of state of the controlled system (and of other events from the controlled system) from one Management-Control entity to another (usually superior) Management-Control entity.

[25] The flow rate and the presence depend upon the need to send information.

In this context, as much of the information is derived from readings of instruments, the flow is often called telemetry[26].

The stream provides engineered flow such that peak load is reduced and spread using some mechanism such as back-pressure and/or selective pruning of detail.

The definition allows for many alternative stream strategies using the same extensible structure. Streaming approaches are defined that:

- Focus on conveying aggregate instances.
    - An aggregate[27] instance is an instance of a global class and all of its contained leaf node instance where each is identified by a local id in the context of the aggregate instance (by an address)[28].
- Enable the client to discover the properties of available streams
- Provide an opportunity for event time reporting that is structured to allow for reporting of time uncertainty (see reference).
- Allow a client to achieve and maintain eventual consistency with the state of the controlled system simply by connecting to the stream(s), i.e., with no need to retrieve[29] current state prior to processing log reports (see also 7.7.9 Fidelity on page 103).
    - Makes it a provider responsibility to send information to the client that ensures eventual consistency is achievable[30], removing sequencing complexity.
    - Improves provider-side scale as simply based on a time-sequenced log of events as opposed to a combination of a time-sequenced log and a repository[31].

  Note: There is still a need for the client to "mark and sweep"[32] to achieve full system realignment on recovery from a major loss of communications.
- Allow for efficient recovery from temporary streaming channel communication failures.
- Take pressure off of a client when under heavy load, recognizing that loss of some detail when under extreme pressure is inevitable and tolerable.
- Remove the need for complex subscription in normal controller-controller interaction.
- Are structured to support augmentation with:
    - Event detail
    - Additional structure

The streaming approach described in this document is aligned in principle with the approach taken by the gNMI community with respect to the operation of the STREAM described in the [GNMI-SPEC].

Streaming can be used in several different applications. The primary application is one where a provider is offering an ongoing flow of state updates to a client.

---

[26] This term loses relevance once the readings have been processed and abstracted but is often still used.
[27] See https://en.wikipedia.org/wiki/Domain-driven_design for more information on the concept of aggregates.
[28]For example, an instance of ForwardingConstruct (global class with its uuid) includes all locally identified branches and leaves (such as FcSwitch and FcPort) but NOT aggregated instances (such as FcRoute, as the FcRoute is a global class separately identifiable using its uuid).
[29] For example, using RESTConf GET.
[30] Clearly the client implementation has to take advantage of this correctly as defined in the relevant use cases.
[31] This removes the need to grab a snapshot of the entire repository to service a full-alignment get and removes the need for an alternative get fragment approach.
[32] A garbage collection strategy (see https://en.wikipedia.org/wiki/Tracing_garbage_collection).

In this application the following assumptions apply:

- The client has one or more internal representations of the semantics (models) of the controlled system (network etc.). A representation may:
    - Relate to a subset of the Core IM (e.g., just physical inventory)
    - Compress or expand parts of the model (e.g., LTPs are expanded into TTPs and CTPs)
    - Be enriched with associations (e.g., some or all of the one-way navigations are converted to two-way navigations)
- The client maintains (stores in some form of repository) an ongoing live view of the relevant states of the instances of elements in the controlled system so as to populate each of its representational forms
    - A mechanism is available that enables the on-going reporting, from the provider to the client, of change in information known to the provider.
      Note: A view that is constructed from the currently known state will necessarily be plesiochronous[33] with respect to the actual network state because of differential network and processing delays. After some period, the temporal inaccuracies can mainly be corrected in a view of a particular past time such that the state that was present at some appropriate past time is determinable. This is consistent with the concept of "eventual consistency".
- When connected for the first time, the client must gain knowledge of current state prior to processing information on change (changes alone are insufficient to provide a clear view of the system state especially recognizing that most states change very rarely – waiting for a change to determine current state is not viable).
    - On connection to the provider, the client gains alignment with the current state (from information in the stream) and then maintains alignment as the state changes
    - Through the on-going process the client populates its repository as appropriate and deals with the challenges of asynchronous receipt (e.g., where the referencing entity arrives before the referenced entity)

Consequently, the provider aims to optimize the process of maintaining alignment for the client.

7.3 Overview of Streaming Characteristics

The key characteristics of the Streaming solution are that it:

1. Ensures "eventual consistency" of the client with the view presented by the provider
    - Essentially, if the controlled system stops changing, once the whole stream has been received and processed in order by the client, the client view will be aligned with the corresponding controlled system state (assuming communication with all components in the controlled system is operating correctly etc.)
2. Is built on a provider log of records detailing change in the controlled system
    - The log is designed to enable "eventual consistency"
3. Guarantees delivery of each log record "at least once"

[33] A plesiochronous system is one where different parts of the system are almost, but not quite, perfectly synchronized.

- Clearly, this guarantee applies within a particular operational envelope as defined in this document such that if communication fails for a long period, then detail of changes is lost. On reconnection, the solution ensures that the client gains "eventual consistency" with the view presented by the provider
- May deliver some information more than once, but this will be in a pattern that ensures "eventual consistency"

4. Is highly scalable and available
    - Boundless scale (with corresponding system resources)
5. Is highly reliable (network fault tolerant)
    - Provides an inherent high availability solution (assuming necessary implementation can be realized on a resilient server). The server can feed multiple instances of client and the client can receive from multiple instances of provider
    - Is tolerant to network communications disruption allowing the client to resume from where it last successfully received a record.
6. Has low latency and high throughput on big data scale
    - Assuming the appropriate implementation technology
7. Offers flexibility in the division of information across streams
    - There can be multiple streams offered by a provider to a client where each stream differs from the others in terms of information content and/or protocol
    - In the case where there are multiple streams offered, the client may have to connect to several streams to get all the information it needs
8. Allows the client to request previously sent records from a given stream any time.
9. Supports back-pressure[34] from client to enable a reactive producer that reduces rate of stream flow dependent upon client capability.

7.4 A compacted log driving a stream

This section works through the rationale for various features of a compacted log and provides a brief view of the alignment process. See also details on compacted logs from [KAFKA-COMP] and description in [ONF TR-548].

7.4.1 Boundless log

The provider will log each change that occurs in an entity in the controlled system by recording the whole entity detail including the new values that result from the change. If an entity is added to the controlled system, the whole entity is recorded. When an entity is removed from the controlled system, a delete event is recorded.

The record about the latest change is appended to the head of the log. The provider can stream the log contents to the client from the oldest record, which relates to the first entity created in the controlled system, and once the client reaches the most recent, the client will be aligned with the current state of the system.

With no further processing the log would grow boundlessly and the alignment time for the client will become longer and longer. This is clearly not a practical approach. The compaction process, which operates on the log in background, prevents this boundless growth by removing records.

[34] Applying some control to reduce the flow from the provider such that the client does not lose information.

7.4.2 The compaction process

In its simplest ideal form, the compaction process will ensure that the log only holds the latest record related to each entity that exists in the controlled system. Hence, when a new change occurs the previous record about that entity will be removed from the log. This removal is the core of the compaction process. When an entity is removed from the controlled system the latest record about that entities state will be removed from the log and a delete record will be appended to the log.

7.4.3 Client connects

When a client connects, the provider will stream from the tail of the log sending the oldest record in the log first. This will probably not be the first record ever appended to the log as it is likely that changes have taken place to the entity related to that original record so that record has been removed by compaction. Of course, it may be the original record, and this may be many years old as the latest record about any particular entity is persistent (a record is only removed through compaction as a result of an update to the state).

As the client proceeds through the log, it will gain a view of the most recent state of each entity in the system. It is likely that as it progresses through the log, changes will be taking place in the controlled system such that a record it has read will be superseded by another record.

This is not an issue as the client will eventually reach that new record and update its state accordingly. This record may be a delete, in which case the client will remove the entity from its view. Once the client has reached the head of the log it will be up to date with the current state of the controlled system. As it will normally be a little behind the provider it will not be quite consistent but will have essentially achieved "eventual consistency" (see section 7.7.8 Eventual Consistency on page 103).

If the client loses connection with the log, it can reconnect and request the stream to start from the record after the one it last successfully processed.

7.4.4 Challenges and further refinement

There are still two obvious challenges:

1. With no further processing the log will grow boundlessly due to an accumulation of delete records.
2. If the client is a little behind on the stream and there is a fleeting change that returns back to the previous state prior to the change, the client may miss this change.

This leads to two timers:

1. Delete retention (also called Tombstone[35] retention), that causes the deletes to be removed from the log after a (long) period of time.
2. Compaction delay, that prevents compaction removing recent records so that the head of the log will maintain a recent change history.

Delete retention creates further challenge in that a client that is a long way behind on the log, receiving records that are older than the delete retention may miss a delete and hence be out of

[35] As Tombstone is a minimal delete message with only the identifier of the deleted entity recorded.

alignment. This leads to the need for the client to realign by going back to the oldest record in the provider log and streaming from there (using a normal "mark and sweep" process to ensure its database does not have deleted records).

7.4.5 Operation

A primary application of streaming is in a solution where a client needs to gain and maintain alignment with a context presented by a provider:

- The provider presents a view in terms of a context and all of its contained instances.
- The client maintains alignment with that view.
- The stream is used for gaining and maintaining alignment with a view.

The next subsections consider the client connection to and receiving from a stream (this is a simplified view of part of the flow described in detail in section 7.7 Operation of the streams on page 96. The description here assumes that the client is capable of consuming the stream at a far greater rate than the stream is being filled. The following provides a brief sketch of the process of gaining and maintaining alignment.

7.4.5.1 Preparing to connect

Once the client has identified the available streams to connect to, the client simply acquires the necessary authorization.

7.4.5.2 Initial connection

On initial connection, the client provides a no record identifier. This causes the provider to stream from the oldest record. The client can continue to consume records from the stream ongoing.

The initial records received by the client will be for the entities that have not changed for the "longest" time[36].

7.4.5.3 Delete retention passed

As the client continues to consume the stream it progresses past the delete retention point, i.e., is receiving records that have a timestamp that is less than the delete retention (delay) point from the current time and recent deletes will be received along with newer changes[37].

Compaction will have removed multiple reports about the same entity, but as the stream progresses further it is possible that an update is received that overwrites previously received entity state or a delete is received that deletes an entity that was read earlier. This is where compaction had not yet removed the entity when the stream was started (potentially because the event causing the newer record had not yet occurred).

7.4.5.4 Compaction delay passed

After some time, the client consumes past the compaction delay point (i.e., is receiving records that have a timestamp that is less than the compaction delay point from the current time). From this point onwards the client is receiving all recent changes and is aligned with controlled system

[36] For example, if the system has been running for three years and a thing was created when the system started. If that thing has never changed since creation, then the record of its creation from three years ago will still be in the log.
[37] Deletes are only retained for a limited time. Delete records older than the delete retention are removed from the log.

as it was perceived by the provider at some recent point in time. Whilst receiving records that are newer than the compaction delay point, the client will receive all event reports for the context.

7.4.5.5 (Eventual) Consistency achieved

If the controlled system stopped changing, then the client would eventually reach the newest record and would be aligned with the provider view of the state of the controlled system[38].

7.4.5.6 Degraded performance

Information fidelity is reduced if the client slips back by more than the compaction delay as compaction will remove some change detail.

7.4.5.7 Need for realignment

If the client processing of a stream is delayed by more than the delete retention[39] such that the backpressure on the stream takes the provider log read point to a record older than the delete retention time, the provider will cause the connection to drop.

When the client detects this, it will reconnect, normally with the identifier for the most recently processed record from the stream. The provider will recognize this as a request for a record older than the delete retention and will stream from the oldest record in the log. This will cause the client to enter into full realignment.

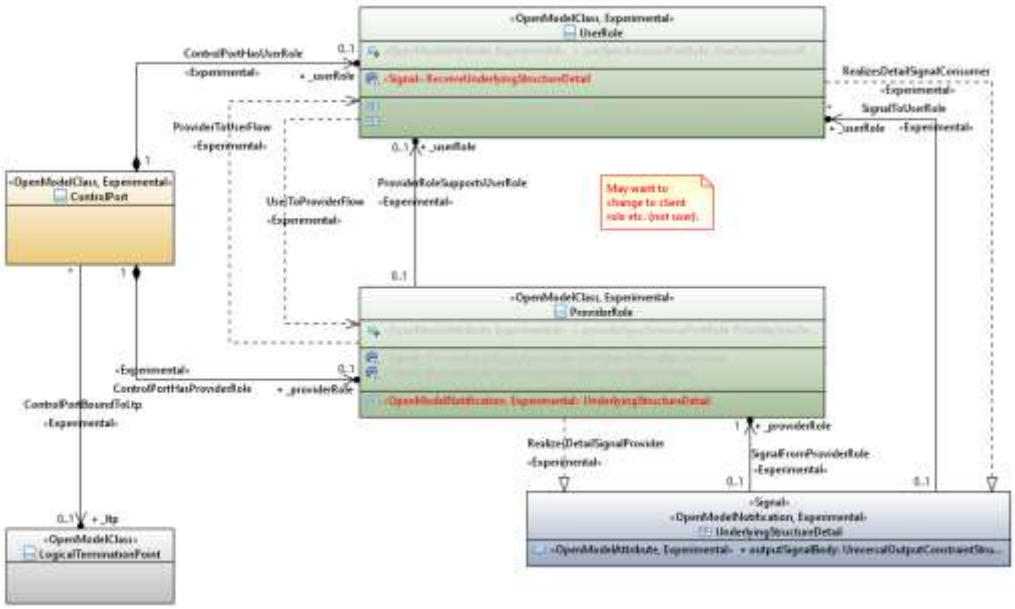
The behavior of the provider at this point is equivalent to that when there is an initial connection. The client may use a "mark and sweep" strategy to minimize the disruption to its view of current state.

7.5 Use of the signal and Get[40]

In section 0

[38] Because the provider logs the whole entity on each change then the most recent record for an entity, retained after compaction has removed earlier records, will include all of its properties.
[39] The TCP buffering will provide some additional time for a client beyond the delete retention.
[40] The ControlPort has operations and signals as it enables system interaction. A client controller talks through a provider control port to the ControlConstruct (and related classes) about the entities being controlled. No other classes in the model have operations, signals etc. as all communication is through the ControlPort.

Operations on page 68, the detailed model of interaction is described. This section highlights the relevant parts of that model for streaming.



CoreModel diagram: Control-InterfaceClassesStreaming

Figure 7-2 The ControlConstruct Streaming interface detail

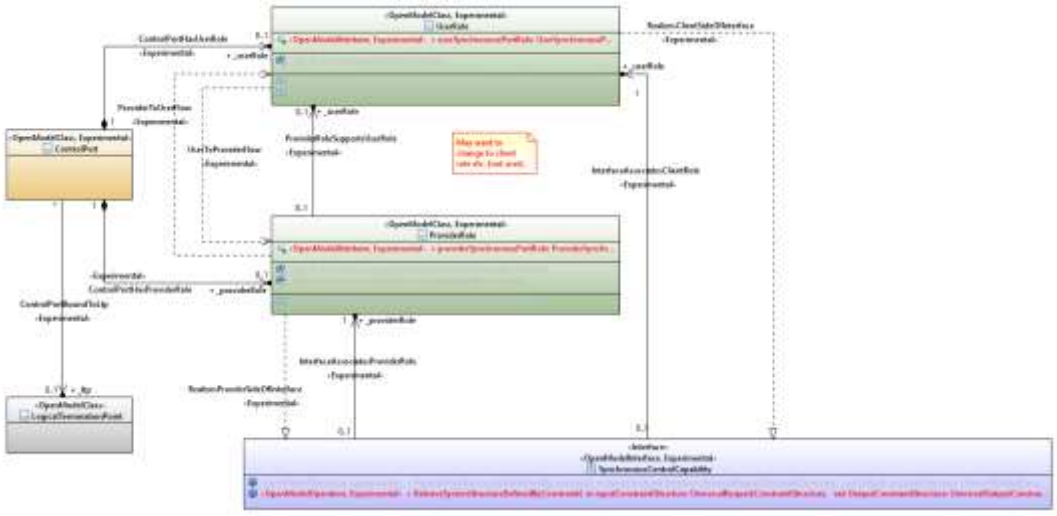
The diagram above shows the relevant detail, drawn from the Operations model, of the structures for the streaming interface with key fields highlighted in red.

The ProviderRole offers the streaming capability and streams "UnderlyingStructureDetail" via the signal highlighted. The signal structure is a universal structure allowing any expression of constraints and specific details.

The UserRole receives the stream, "ReceiveUnderlyingStructureDetail", through the ProviderToUser flow and via the signal receiver highlighted.

The stream operation is covered in detail in the following sections.

The stream also requires use of some basic Get operations. The Get operation is supported by the structures highlighted in red in the diagram below.



CoreModel diagram: Control-InterfaceClassesGet

Figure 7-3 The ControlConstruct Get interface detail

The Get operations are discussed in the context of stream provider capability discovery.

7.6 The streaming model

The streaming model is considered both from the perspective of the provider and from the perspective of the client.

The provider may support more than one stream. If more than one stream is supported the set of streams will include all of the information that the client requires. Each stream will provide a subset of the information that is for a specific purpose e.g. physical inventory; fault management; connection management. A client is expected to connect to the streams that contain the information that the client requires.

7.6.1 Stream provider capability description

The provider supports various streams and can inform the client, on request, of the details of these streams.

It is assumed that the provider has used some policy etc. to determine what streams to support and has initialized those streams appropriately. Stream initialization involves instantiation on the provider of:

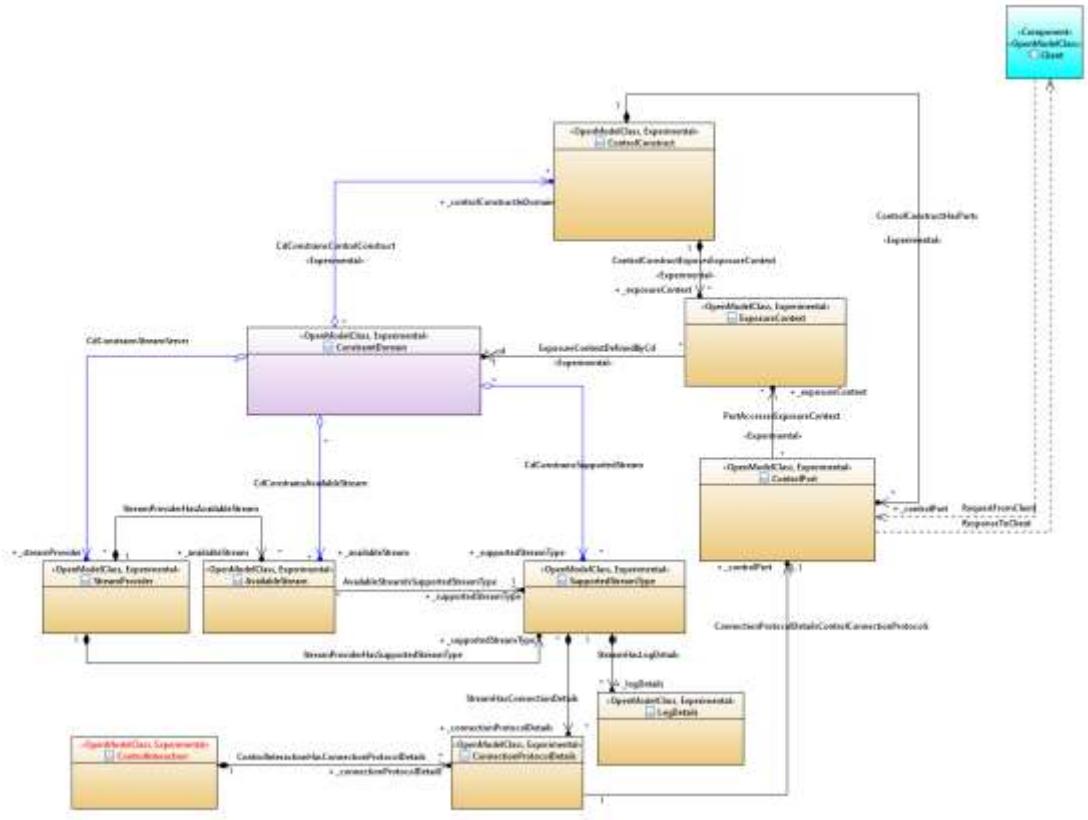
- An appropriate ExposureContext that will provide the view to be streamed (as discussed earlier in this document)
- An associated ConstraintDomain that will contain the relevant entities of the view (as discussed earlier in this document)
- A ViewMappingFunctioin (may be several) that will map from the internal entities (administered and controlled by the provider solution) to the entities of the view required by the client (as discussed earlier in this document)
- A TransmitStreamPipeline and associated entities to support the stream (discussed later in following sections)

- The various capability entities discussed in this section

The provider will initialize the relevant streams ready for client connection. The streams may go through a lifecycle which may include initialization where the stream will be aligned with the current state and include normal active operation where the stream is available for connection. It may also be possible for the provider to pause a stream etc.

Prior to connecting, and outside the scope of this model, the client would be expected to deal with any necessary authentication and authorization. This may result in the acquisition of a token that will be used during interactions.

The figure below shows the key parts of the provider model related to expression of capability. The client will get the information on the supported and available streams prior to connecting to any streams. The process of getting the information related to the streams is discussed later.



CoreModel diagram: Control-WithLogAndStream-AvailableStreams

Figure 7-4 Available streams

7.6.1.1 StreamProvider

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::StreamProvider

The entity that provides access to stream capability information.
This class is Experimental.

Table 12: Attributes for StreamProvider

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_availableStream	Experimental	See referenced class
_supportedStreamType	Experimental	See referenced class

7.6.1.2 SupportedStreamType

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::SupportedStreamType

Definition of a type of stream that is supported by the provider.
This class is Experimental.

Table 13: Attributes for SupportedStreamType

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
streamTypeName	Experimental	Name of the stream type.
recordRetention	Experimental	Time in minutes. Statement of retention time and/or retention capacity in bytes. Key word "FOREVER" means that records will never be removed from the log. May be overridden for particular cases of specific LogStorageStrategy (via augment). Applies to all record types in the stream unless overridden by another parameter (such as tombstone retention for a compacted log).
segmentSize	Experimental	Size of sub-structuring of the log.
logStorageStrategy	Experimental	Indicates the storage characteristics of the log supporting the stream.
logRecordStrategy	Experimental	Indicates the type of content of each log record.
recordTrigger	Experimental	Defines the trigger to log a record.
streamTypeContent	Experimental	Identifies the classes that are supported through the stream. The list may be a subset of the classes within the context.
_logDetails	Experimental	See referenced class
_connectionProtocolDetails	Experimental	See referenced class

7.6.1.3 AvailableStream[41]

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::AvailableStream

Details of a stream that can be connected to by a client application.
This class is Experimental.

Table 14: Attributes for AvailableStream

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
connectionAddress	Experimental	Provides the address for the connection. The format of the address and attachment mechanism will depend on the connection protocol defined in another attribute of this class. There may be a sequence of operations required, in which case, these should be listed as separate strings. A string may include wildcard substatements. A single string may list alternatives separated by an appropriate delimiter.
streamState	Experimental	The state of the stream.
streamId	Experimental	The id of the stream (alternative to the uuid).
connectionProtocol	Experimental	Names the connection protocol for this particular available stream. The connection protocol is chosen from the list of connection protocols identified in the referenced SupportedStreamType.
_supportedStreamType	Experimental	See referenced class

7.6.1.4 LogDetails

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::LogDetails

Details of the log methods available for the specific stream type.
Property examples given for a compacted log.
This class is Experimental.

Table 15: Attributes for LogDetails

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description

[41] There may be many supported streams only some of which may be available, i.e., running, as running uses resources.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
tombstoneRetention	Experimental	Time in minutes. The time period for which a Tombstone record will be held in the log from when it was logged. This provides an adjustment to the essential Compaction strategy such that after the tombstoneRetention period there will be no records about a particular thing that existed but no longer exists. Tombstone retention overrides recordRetention for Tombstones. Key word "FOREVER" means that Tombstone records will never be removed from the log. Can be adjusted by an administrator (via a separate view) through the life of the stream.
compactionDelay	Experimental	Time in minutes. The delay between logging the record and making the record available for compaction. This provides an adjustment to the essential Compaction strategy such that there may be several distinct records for the same thing in the where those records are not older than the Compaction Delay. Can be adjusted by an administrator (via a separate view) through the life of the stream.
maxAllowedSegmentRollDelay	Experimental	The maximum time the log head segement can be allowed to be not made available for compaction. Applicable where the log is segmented and the head segment is not available for compaction. The setting influences the compaction behavior and may cause a delay before compaction that is much greater than the defined compaction delay. Time in seconds. Can be "FOREVER". Can be "NOT_APPLICABLE" (which indicates that compaction can act on the head segment).
maxCompactionLag	Experimental	The maximum delay, in seconds, beyond the defined compaction delay for compaction processing to take place. May be "NOT_APPLICABLE" if compaction is essentially immediate (i.e., there is negligible delay).
_streamLog	Experimental	See referenced class

7.6.1.5 ConnectionProtocolDetails

Qualified Name:
CoreModel::GeneralControllerModel::LogAndStream::ConnectionProtocolDetails

Details of the connection protocols and encoding formats available for the specific stream type.
This class is Experimental.

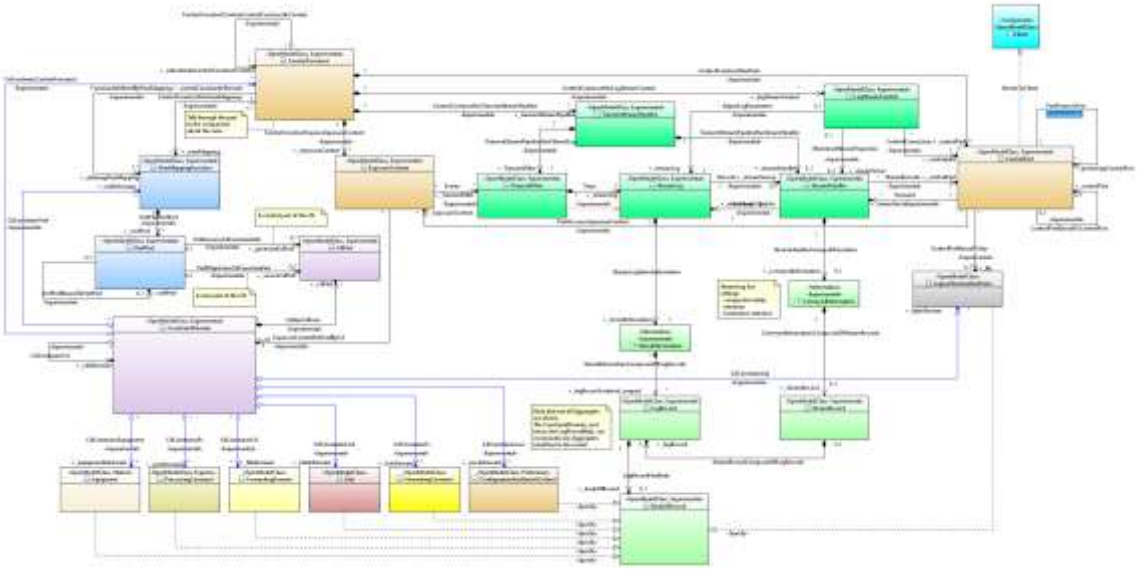
Table 16: Attributes for ConnectionProtocolDetails

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
allowedConnectionProtocols	Experimental	Name of the allowed protocol(s). Where there is a list: - all protocols must use the same encoding format - there will be one or more available streams per connection protocol CONDITION: Mandatory where not default.
encodingFormat	Experimental	The encoding format of the streamed records. CONDITION: Mandatory where not default.
_controlPort	Experimental	See referenced class

7.6.2 Stream provider streaming function

The stream provider will populate the streams with appropriate records depending upon the mode of operation of the stream. For a compacted log stream the stream will essentially hold current state and recent change.



CoreModel diagram: Control-WithLogAndStream-Provider

Figure 7-5 Stream Provider

The diagram above shows aggregate roots such as ForwardingConstruct. It is important to recognize that the aggregate root instance and all of its aggregate leaf instances are streamed as one unit. So when, for example, a ForwardingConstruct aggregate instance is created, the ForwardingConstruct instance along with its FcPort instances, FcSwitch instances etc. will be streamed as a single record. When there are changes to an FcPort instance of the ForwardingConstruct aggregate instance, this will be streamed as part of the ForwardingConstruct aggregate instance (as a single unit).

See also 7.7.1 Provider initializes streams on page 97.

7.6.2.1 TransmitStreamPipeline

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::TransmitStreamPipeline

Set up by ControlConstruct. Relates to a particular flow of information from the ExposureContext
Controls a specific stream integrity (using sequence numbers etc.)
Coordinates the TransmitFilter settings for the stream.
Coordinates the StreamHandler.
This class is Experimental.

Table 17: Attributes for TransmitStreamPipeline

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_TransmitFilter	Experimental	See referenced class
_streamHandler	Experimental	See referenced class

7.6.2.2 TransmitFilter[42]

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::TransmitFilter

Defines which events are sent to a specific Topic.
The expectation is that ALL events from an ExposureContext will be available through the combination of TransmitFilters and streams such that a client connected to an appropriate set of streams can maintain alignment with the entire ExposureContext as defined by its ConstraintDomain.
Provider defines one or more profiles defining a set of streams all of which provide full ExposureContext access and the user selects a profile. May be by negotiation, but certainly not created by the client on the fly with random choices.
If the desired profile does not exist then a slow timeframe process will cause it to become available in the next release of the product/catalogue/etc.
This class is Experimental.

Table 18: Attributes for TransmitFilter

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_streamLog	Experimental	See referenced class

[42] May want to allow the client to specify the streams but there is a danger that the client does not set up the right combination of streams to ensure that all content is available. Maybe a set of profiles? Based upon orchestrator internal partition.

7.6.2.3 StreamLog

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::StreamLog

Stores StreamLogRecords for a particular Topic.
May do compaction/truncation etc. May use a technology such as Kafka.
This class is Experimental.

Table 19: Attributes for StreamLog

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_streamSource	Experimental	See referenced class
_storedInformation	Experimental	See referenced class

7.6.2.4 LogRecord

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::LogRecord

Definition of a record in a StreamLog.
These records are immutable. Once logged they will never change (but can be removed from the log as appropriate).
Includes log record header information.
- record type (create, delete etc.)
- record token and record sequence number
- time the record was captured in the log
- source authenticity token

This class is Experimental.

Table 20: Attributes for LogRecord

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
token	Experimental	A coded (and compact) form of the fullLogRecordOffsetId. This property is used to request streaming from a particular point (e.g., the last correctly handled record). For a basic log solution this may simply be the sequence number.
fullLogRecordOffsetId	Experimental	This property must minimally provide a logging sequence number. Note that when compaction is active, the streamed sequence may not have sequence numbers that simply increment by one. In a complex log solution there may be various parts to the log. The record token is a compressed form of log record reference. This property provides the verbose form For example, it may include: - stream id - topic - partition - partition offset - sequence number (the offset is essentially the sequence number associated with the partition)

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
logAppendTimeStamp	Experimental	The time when the record was appended to the log.
entityKey	Experimental	The identifier of the entity that is used in a Compacted log as the compaction key. The entityKey value, where appropriate, may be based upon the identifiers from the event source. It can be built from some specific detail combination that meets the necessary uniqueness and durability requirements. entityKey is the value used during compaction. Ideally it is a UUID format, if this can be formed from the source identifier.
recordType	Experimental	The type of the record. Can be used to understand which elements of the record will be present.
recordAuthenticityToken	Experimental	A token generated using a method that allows the client to validate that the record came from the expected provider.
_bodyOfRecord	Experimental	See referenced class

7.6.2.5 BodyOfRecord

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::BodyOfRecord

The key contents of the log record. Details for one Aggregate instance.
- event time (allowing for time inaccuracy)
- aggregate type
- identifier of the containing parent aggregate

Note that the Aggregate will reference other Aggregate instances some of which may be "behind" this record in the log.
This class is Experimental.

Table 21: Attributes for BodyOfRecord

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
eventTimeStamp	Experimental	Time of the event at the origin of the event that triggered the generation of the record. The structure allows for time uncertainty.

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
parentAddress	Experimental	Where the entity to be reported is a global class this provides the compositional position of the entity in terms of an ordered list of UUIDs. Where the entity is a local class this provides the ordered list of ids through composition via the next parent which may be a local class to the closest global class (which may be the next parent). The field can then also include ids of all entities back to the Context and hence can be used for global classes where the tree is being represented in full. Gives the position of the entity in the address tree (usually containment) that is raising the event by providing the name/id values in the address of the parent. Is the sequence of named levels in the tree up to but excluding the entity of the notification. A stream record may include only a portion of a DDD entity where only that portion has changed.
recordContent	Experimental	The identifier of the object class in the record body detail. This property is used to control the conditional augmentation of the body with detail.

7.6.2.6 LogStreamControl

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::LogStreamControl

For stream transmitter, monitors StreamLog state and StreamHandler behaviour.
For stream receiver, monitors ReceiveStreamBuffer.
Controls the communications in response to monitored conditions dropping and restarting connections as appropriate.
Need to explain how the client chooses the record to start from.
This class is Experimental.

Table 22: Attributes for LogStreamControl

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_controlport	Experimental	See referenced class
_streamLog	Experimental	See referenced class
_streamServer	Experimental	See referenced class
_receiveStreamPipelineBuffer	Experimental	See referenced class

7.6.2.7 StreamHandler

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::StreamHandler

Represents the function that maintains flow integrity of a stream.
This functions

- feeds StreamRecords to the ControlPort built from StreamLogRecords from the StreamLog.
- responds to backpressure from the ControlPort accounting for Demand.
This class is Experimental.

Table 23: Attributes for StreamHandler

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_controlPort	Experimental	See referenced class
_streamLog	Experimental	See referenced class
_conveyedinformation	Experimental	Information conveyed by the stream handler.

7.6.2.8 StreamRecord

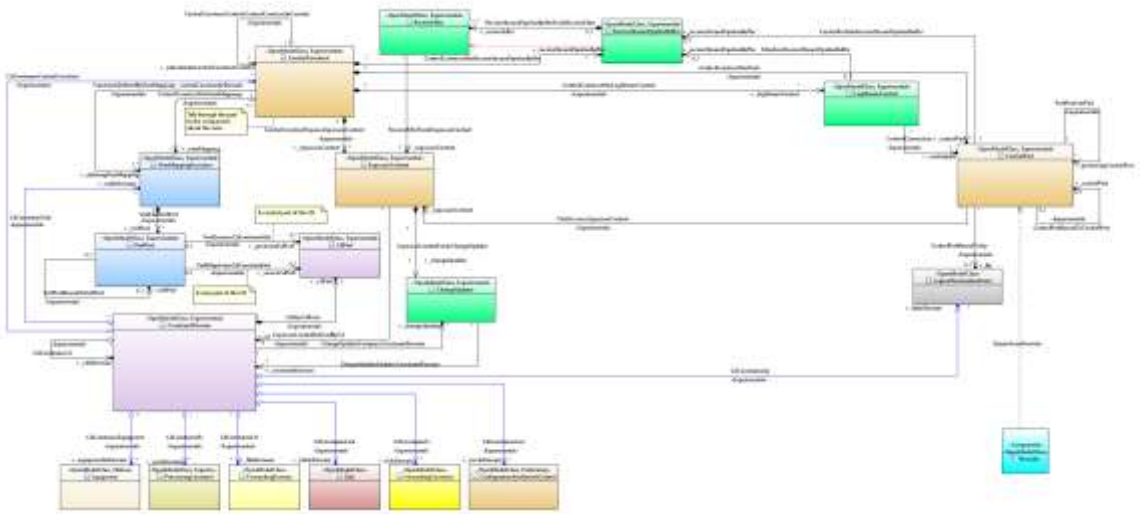
Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::StreamRecord

The record sent in the stream.
Includes:
- record identifier
- time the record was formed
Note that this record may contain records for multiple Aggregate instances of mixed Aggregate types.
This class is Experimental.

Table 24: Attributes for StreamRecord

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_logRecord	Experimental	See referenced class

7.6.3 Stream client



CoreModel diagram: Control-WithLogAndStream-Client

Figure 7-6 Stream Client

The diagram above shows all key client functions that control the port, buffer the flow, filter the flow (minimal coarse filter) and deal with updating the context with changes.

See also 7.7.3 Client connects to a stream on page 99.

7.6.3.1 ReceiverStreamPipelineBuffer

Qualified Name:
CoreModel::GeneralControllerModel::LogAndStream::ReceiveStreamPipelineBuffer

Buffer that balances flow from the communications and storage rate differences.
Applies backpressure on the communications system. Forms part of the overall stream pipeline.
This class is Experimental.

Table 25: Attributes for ReceiveStreamPipelineBuffer

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_receiveFilter	Experimental	See referenced class
_storedInformation	Experimental	See referenced class

7.6.3.2 ReceiverFilter

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::ReceiveFilter

Filter that removes stream records that are not relevant to a specific receive ExposureContext or ControlConstruct (for control messages).
This class is Experimental.

Table 26: Attributes for ReceiveFilter

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_exposureContext	Experimental	See referenced class
_controlConstruct	Experimental	The receive filter will route messages to the control construct where the message contains a request as opposed to information to apply to the exposure context.

7.6.3.3 ChangeUpdater

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::ChangeUpdater

Function that recognizes difference between the current state of an Aggregate (in the related ConstraintDomain) and the Aggregate details in the record received and causes necessary updates to the Aggregates in the related ConstraintDomain.
This function deals with Idempotent behavior.
Removes properties the client is not interest in.
This is intentionally a limited capability that can simply remove some properties and not perform major transformations or complex filters.
This class is Experimental.

Table 27: Attributes for ChangeUpdater

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_constraintDomain	Experimental	See referenced class

7.6.3.4 RequestConstructor

Qualified Name: CoreModel::GeneralControllerModel::LogAndStream::RequestConstructor

Function that coordinates requests originating from the ControlConstruct relates to specific contents of an ExposureContext and sends the request to the appropriate ControlPort.
This class is Experimental.

Table 28: Attributes for RequestConstructor

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
_controlPort	Experimental	See referenced class
_exposureContext	Experimental	See referenced class

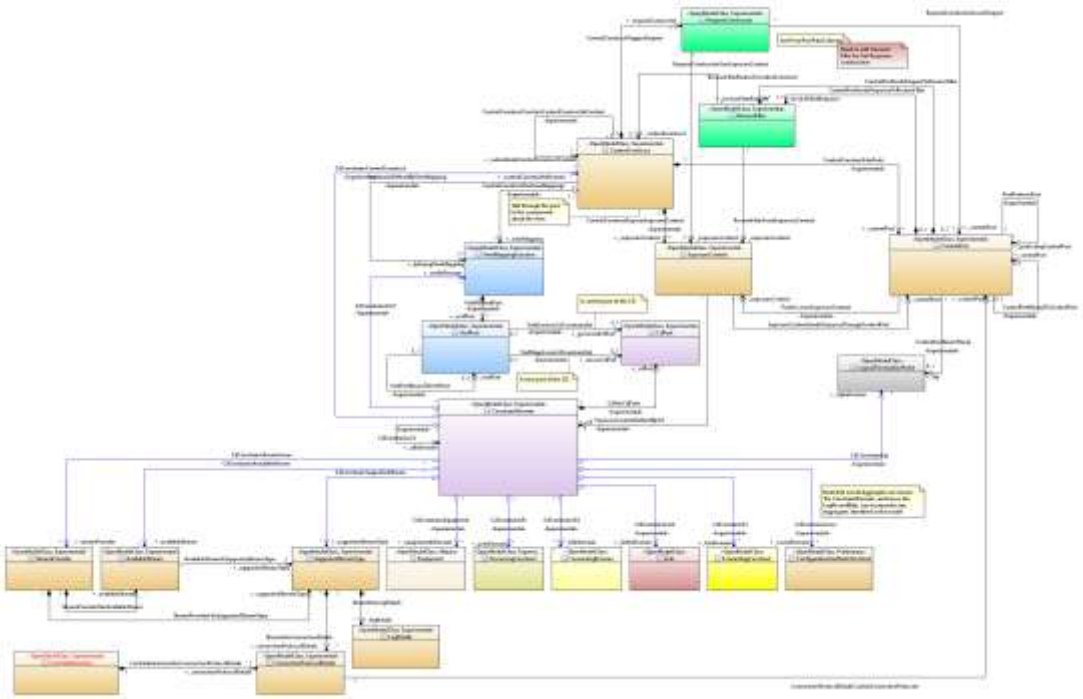
7.6.4 General request constructor and Get request

The following diagram shows the key classes for construction of a Get and processing of the response. Both the provider and client parts are shown together.

The solution provides a simple Get function where the client requests information that is maintained in the ConstraintDomain, aligned with the source via appropriate ViewMappingFunctions and accessed directly via the ExposureContext. This is the same information that is streamed on change.

The solution also offers a deep Get function where the information requested by the client is NOT directly available and where the ControlConstruct triggers an instantaneous mapping that acquires the data from deeper in the system, potentially the fundamental source (via a recursion of such ViewMappingFunctions) and then provides a fleeting view via the ConstraintDomain and the get[43].

A similar function is supported for the snapshot stream where the client operation triggers the streaming of a fleeting representation of data that is NOT maintained live in the ConstraintDomain associated with the ExposureContext[44].



CoreModel diagram: Control-WithLogAndStream-Get

Figure 7-7 Get

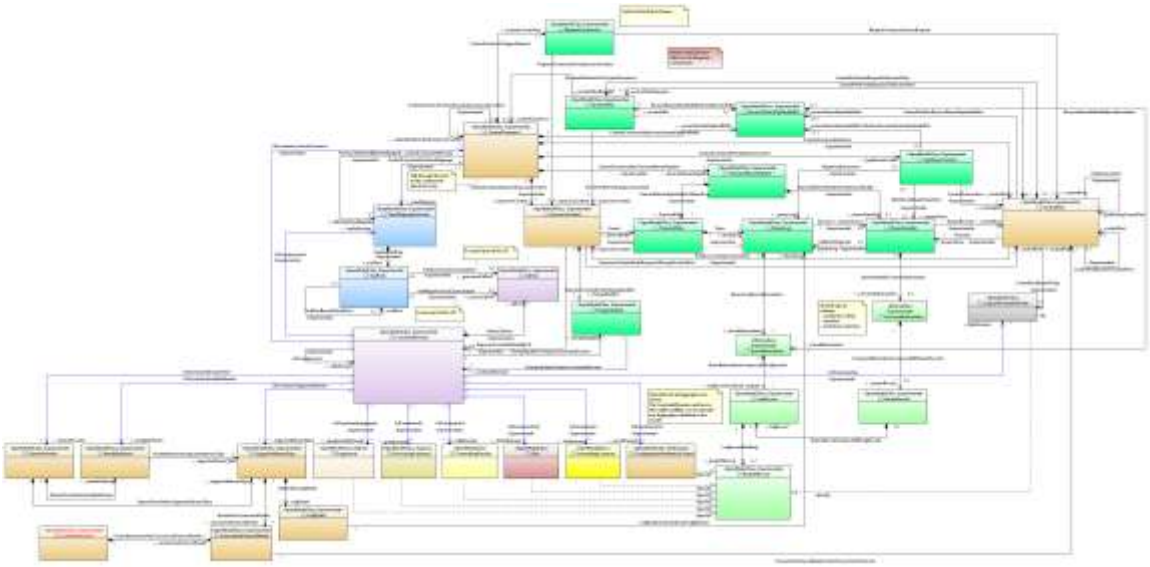
[43] The client becomes aware that the information is available via an expression of capability exposed by the provider. The CD content is a collection of instances. The capability is a set of occurrences (instances at the next higher descriptive level).

[44] There is no need to get as the information is streamed. This allows for asynchronous and delayed responses etc.

7.6.4.1 RequestConstructor

See 7.6.3.4 above.

7.6.5 Complete model



CoreModel diagram: Control-WithLogAndStream-Complete

Figure 7-8 Stream Model

7.7 Operation of the streams

The figure below shows a control hierarchy (right) with a device at the bottom with its controller above it and an orchestrator above that. The expanded view explains the symbols and show an arrangement of control entities (each discussed in this document). The Network View is any collection of model entities from the Core Model (as depicted in this document).

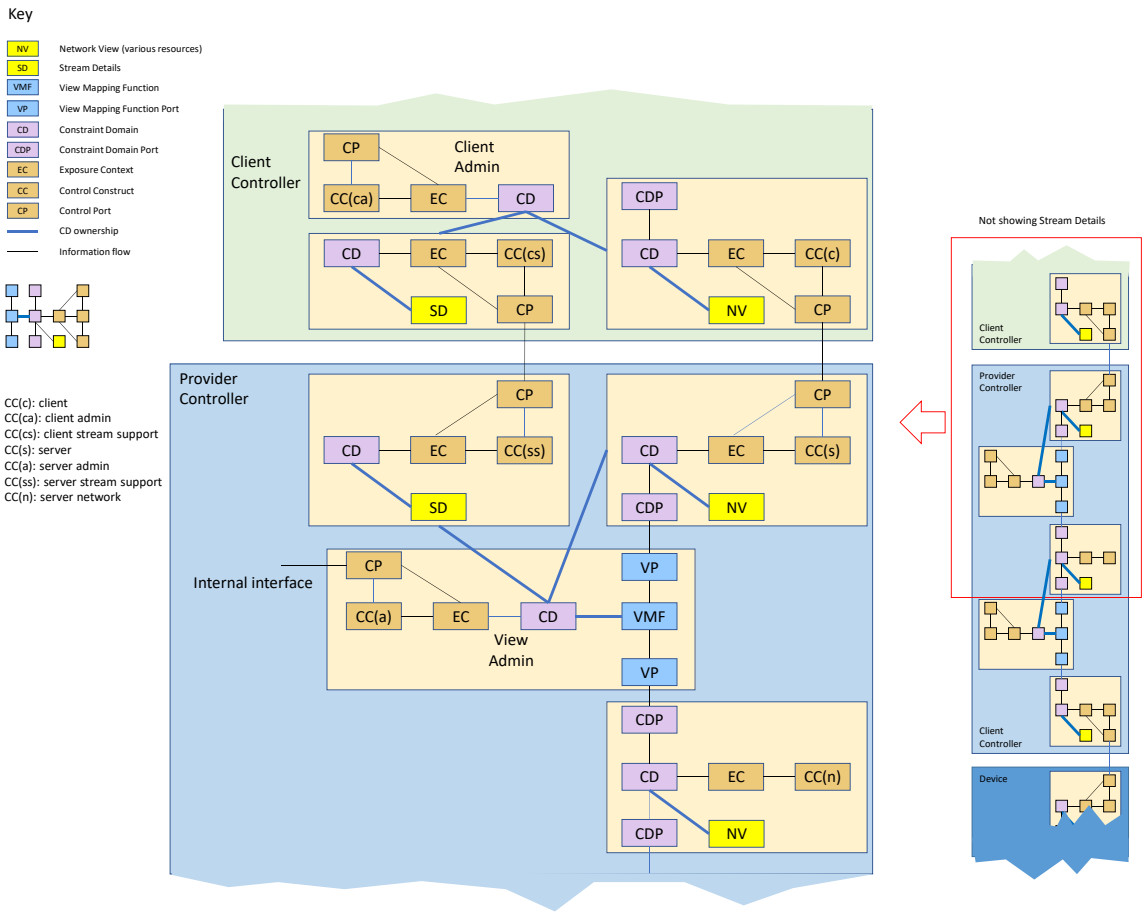


Figure 7-9 A controller hierarchy in detail

The following subsections run through a progression of lifecycles of the provider and the client. The text assumes that the provider sets up the streams first, then the client discovers the streams and finally the client connects.

Clearly, the provider could add streams on the fly for the client to discover etc.

Determining which specific views are to be presented and creating the VMFs etc. is outside the scope of this discussion. It is assumed in this discussion that the VMFs are in place etc.

7.7.1 Provider initializes streams

Considering Figure 7-9 A controller hierarchy in detail on page 97:

1. CC(s) is instantiated and configured, by CC(a) using policy/specification/profile detail provided to CC (a) using a mechanism not discussed here.
    a. CC(s) will be configured to support a single EC with an associated CD.
    b. CC(a) will also instantiate a related VMF where that VMF has been configured by CC(a), using policy/specification/profile, to map from some native classes of the Controller handled by CC(n) to ONF Core Model classes to be exposed.
2. At initialization of the CC(s), the VMF will run to construct the appropriate view.

    a. That view is collected by the CD (shown as NV) in CC(s) and includes a set of instances of ForwardingConstruct etc.
    b. Note that the VMF has been defined and created by a function outside the scope of this discussion. This function would be realized in terms of CCs etc.
3. Assuming that the CC(s) is configured to support streaming, there will be an associated view of the streaming provided.
    a. This view is made available via CC(ss) which will advertise SupportedStreamTypes and AvailableStreams. Not all SupportedStreamTypes advertised will be available.
        i. Some policy/specification/profile will inform the CC(a) of which of the SupportedStreamTypes should be made available.
        ii. CC(a) will configure CC(ss) and CC(s) appropriately (the explicit mechanisms are not discussed here).
4. For each AvailableStream CC(s) will create a corresponding TransmitStreamPipeline with a compatible TransmitFilter feeding an associated StreamLog.
5. Assuming that the streams are based upon compacted logs, at initialization of the stream CC(s) will trigger its EC to populate the StreamLog with the current state of the relevant aggregates from NV via the CD, refined by the appropriate TransmitFilter.
    a. This process of populating the StreamLog will be governed by the TransmitStreamPipeline.

At this point the provider is ready for client connection.

7.7.2 Client gets capability information related to the provider streams

Considering Figure 7-9 A controller hierarchy in detail on page 97 and assuming that "Provider initializes streams" has run to completion:

1. A Client Controller has been directed to use a particular CC(cs) for interface administration.
2. CC(cs) is provided with information on streams that may be handled by CC(c).
    a. The specific mechanism for this set up is not shown but would be similar to that associated with CC(a).
3. CC(cs) is also informed of the identity of a Provider Controller that it is to connect to request stream details.
    a. It has appropriately compatible connection support to connect to that Provider Controller.
4. In preparation to deal with the response from the request, the CC(cs) sets up an appropriate ReceiveFilter associated with its CD.
    a. A Task coordinates this work.
    b. The ControlConstruct and any necessary PCs interwork to achieve this setup (the details are not discussed here).
5. CC(cs) sends a request for AvailableStream and SupportedStreamType (using a GET method).
    a. Prior to sending the request the client will have had to have been authenticated and perhaps acquired a security token that it will pass with the request. A Task coordinates this work (the detail of which is not discussed here).

6. The Provider Controller receives a request for connection (via an LTP associated with the CP of CC(ss)) and when connected receives a GET request.
    a. This GET is directed by the ReceiveFilter to the CC(ss) which interprets the GET.
    b. The GET is forwarded to the CD which further interprets the GET.
    c. The CD triggers the EC to provide response details through the CP.
    d. The details of the supported streams are returned.
        i. This includes the connection method (connectionAddress)
7. The client receives the response through the CP of CC(cs) and this is passed to the ReceiveFilter which passes it to the EC and is then built in the CD to retain information on SupportedStreamTypes etc.

At this point the client has all necessary detail to enable it to connect to the provider streams.

7.7.3 Client connects to a stream

Considering Figure 7-9 A controller hierarchy in detail on page 97 and assuming that "Client gets capability information related to the provider streams" has run to completion so that the client has discovered AvailableStreams etc. from the Provider Controller.

1. CC(ca) coordinates the selection of a relevant stream from the AvailableStreams and reads the connection method details etc.
2. CC(ca) directs CC(c) to use the connection method to connect to the stream offered by CC(s) and to ask for records from the oldest.
3. CC(s) receives a request for connection (via a CP) and associated ReceiveFilter.
4. CC(s), running a suitable Task, validates the request and, assuming valid, creates a LogStreamControl function associated with the relevant CP and StreamLog
5. CC(s), via the appropriate TransmitStreamPipeline function instance, creates a StreamHandler.
    a. CC(s) associates the LogStreamControl with the StreamHandler.
6. The stream is then started by the TransmitStreamPipeline function.

At this point the provider is starting to stream to the client

7. The LogRecords are formed into StreamRecords and sent, as dictated by the flow control of the TransmitStreamPipeline, through the CP via the Signal source UnderlyingStructureDetail.
8. The StreamRecords are received via the Signal sync ReceiveUnderlyingStructureDetail of CP of CC(c) and transferred from the CP to the ReceiverStreamPipelineBuffer.
    a. The ReceiveStreamPipelineBuffer has StoredInformation of the form of StreamLogRecords.
9. The stream sink feeds an EC in the client.
    a. The EC is being used to feed the CD (reverse of the source arrangement discussed in 7.7.1 Provider initializes streams on page 97).
    b. The stream pipeline of CC(c) has a ReceiveFilter that feeds the EC.
    c. There may be several ECs on a stream.
    d. Not all entities from a stream are applied to a particular EC.
    e. Not all of an entity from a stream may be applied to a particular EC (i.e., only part of the entity is stored).

f. The stream sink has flow control.
g. The sink can apply back pressure on the stream, and this is essentially a pipeline backing up to the StreamLog in CC(s).
h. The stream pipeline should have integrity up to the sink persistent store in the CD of CC(c).

10. The ChangeUpdater identifies changes and updates the contents of the CD appropriately.

The sequence 7-10 is a continuous process with each steps running repeatedly in parallel providing an ongoing flow.

Assuming that the provider is using a compacted log the client will achieve "Eventual Consistency" with the provider as the process above continues.

Considering the compacted log, the stream flow 7-10 will start with records identifying current state. If no changes occur during the client progression through the stream, then the streamHandler in the provider will reach the head of the log and the client will then be aligned with current state.

During the progression of the stream, it is highly likely that some changes will occur in the controlled system. So as the client proceeds to consume the stream it will find the current state mingled with change.

Assuming that the system is very large and hence the current state takes significant time to absorb, changes will work their way through the log and may move into the compaction zone prior to the client stream reaching the record of the changes.

If further changes occur to the entity that has records in the compaction zone, these records will be compacted (i.e., be removed from the stream), and the client will not be aware of that particular change detail. This is not an issue as the intention is that the client achieves eventual consistency. Once it has moved from the compaction zone it will be receiving all changes, so will achieve full fidelity and when it reaches the head of the log, the view the client has of the state of the underlying system will be aligned with the state of the underlying system.

When the client is running normally, close to the head of the log and outside the compaction zone, steps 7-10 will be conveying latest changes.

7.7.4 Aggregates in detail

Considering the provider (CC(s))

- Aggregate instances augment BodyOfLogRecord instances
  - An Aggregate is a root (a class that is not contained) and its leaves (contained classes).
- Create of Aggregate (e.g., LTP, FC) is simple augment
- Update[45] can be "whole Aggregate on change" or "change only"
  - "whole Aggregate on change" is simple augment for create and update

[45] If there is a change in the core of the controller in a forwarding structure that is to be exposed as an FC, that change is exposed by the corresponding internal EC and will propagate to a VM which will map to the corresponding FC which will be changed. This change will be exposed through the EC and propagated to the stream. A client connected to the stream will see the change when it is at that point in the log.

        - An explicit delete may only need the identifiers
    - "change only" requires
        - Only the changing Leaf is sent, in the context of the aggregate internal tree structure, stating only the change
        - Aggregate structure to essentially have all non-identifying properties and each leaf as optional (so that only the changed properties are in the log)[46]
            - Each optional Leaf/property is essentially conditional on relevance
        - A way of expressing addition to structure (add, append, insert) and removal from structure (relates strongly to delete) as well as rearrangement of the structure
        - A way of deleting an Aggregate which be the same as explicit delete for "whole Aggregate on change"
    - "Change only" may benefit from a way of mixing additions and deletes in a single expression
    - An aggregate may have interrelated internal properties
    - Any update to the interrelated properties must be reported such that the Aggregate maintains internal consistency
        - For example, if property A changes to 5 and this forces property B to change to 3, both changes must be reported in one single message
- The identification method may use an address (sequence of ids)

7.7.5 Stream communication issues

If there are issues with the Stream detected by LogStreamControl of CC(s), it causes the CP of CC(s) to drop the appropriate connection. The CC(c) detects the connection has dropped and initiates a reconnection using the appropriate connection method at its CP. It then asks for records beyond the last one it successfully processed (recorded by its CD).

The provider receives a request for connection (via a ControlPort) and this is directed by the ReceiveFilter to the appropriate (addressed) ControlConstruct. The ControlConstruct (running a suitable Task) validates the request and, assuming valid, causes LogStreamControl to assess whether it is appropriate to continue to stream from the last record successfully received by the client or whether it is appropriate to restart the stream. The stream pipeline is started appropriately (as described in section 7.7.3 Client connects to a stream on page 99).

Where the stream is restarted by the provider the client will need to run the appropriate realignment process as described in section 7.4.5.7 Need for realignment on page 80.

The client may detect disruption to the stream due to communications failure or some loss of integrity of the flow (such as discontinuous sequence numbers). The client may choose to drop and reconnect from an appropriate point.

[46] All leaves and all non-identifying properties should be optional in the transfer syntax such that any properties can be omitted, and the message still be conformant with the syntax. Hence a single changed property can be reported in the context of an identified tree structure of the aggregate. Each property will be required under particular circumstances. This will be expressed by a condition statements which will contain the rules. Hence the properties are all semantically conditional mandatory. This also applies to multiplicities.

7.7.6 Asynchronous streams

A controller in any control solution will be faced with the challenge that, regardless of the mechanism used to acquire information about the controlled system:

- Knowledge of the state of the controlled system at the controller will be delayed:
  - Hence the controller will never have knowledge of the current state of the controlled system, its information will always be slightly historic.
- Information from the controlled system will not necessarily arrive in the order it occurred, and it will be skewed in time.
  - As a result, there will be temporary unresolvable references in the controller view.
  - For example, it is possible that an aggregate instance that includes a reference to another aggregate may appear in the stream prior to the arrival of the aggregate that it referenced.

Due to this fundamental asynchronous nature of any real solution space and the skewed time of arrival of information, an eventual consistency approach has been chosen.

Clearly, the client solution will need to be able to deal with partially consistent inter-aggregate relationships and with the "gradual" emergence of consistency/clarity.

If changes are happening fast enough, the controller may not get to full consistency/clarity regarding one change before another change occurs resulting in further temporary inconsistency (blurs the view).

If a controller is maintaining a full history of received information and that information carries an accurate timestamp from the source for each event, after some time elapses the controller will be able to form an accurate view of the history. For example, if the controller waits for an hour after some events and then arranges the events in the chronological order of their occurrence (instead of the order in which they were received), it will be able to construct an accurate view of the changing state of the controlled system and its behavior overtime.

7.7.7 Event time accuracy

Accurate event time recording at the origin of the detection of the event is necessary for successful interpretation.

Issues of time precision, time synchronization and determination of "event moment" can reduce the quality of the interpretation of the controlled environment.

It is beneficial for the solution to support the opportunity for:

- The source of the event to indicate time accuracy and to express how it determined "event moment".
  - This may be via a spec describing the detection process.
- An intermediate system that suspects time issues, e.g., the event report indicates an event time that is in the (near) future, or the time information is missing.
  - This may be via a specific structure in the report giving an opportunity to report approximate time.

Issues with time accuracy will degrade the control solution decision making.

7.7.8 Eventual Consistency

A simple way to understand eventual consistency is to imagine a network that is changing such that there is a stream of changes and where the client has not absorbed the stream. If suddenly the network were to stop changing, then, once the client has absorbed the entire stream, the client will be aligned with network state.

7.7.9 Fidelity

In this document, fidelity is the degree of exactness with which the controlled system behavior is represented via the stream.

Loss of fidelity is the loss of some details of change (without losing eventual consistency). For example, the operational-state property of an LTP may initially be ENABLED, then become DISABLED and then ENABLED again. Loss of fidelity may lead to the operational-state being observed (by the client) as continuously ENABLED.

7.7.10 Visibility

It is possible for information on the controlled system to become permanently lost. For example:

- Due to failure of a control component where there is no resilience
- Due to resource limitations in the stream buffering

The streaming solution should be such that the client is able to determine that there has been a loss of information due to limitations in the streaming solution itself.

7.8 Get/Post/Put/Patch

Traditional solutions use request methods such as GET to acquire information and a POST/PUT/PATCH to apply changes.

The model caters for these traditional methods, but also allows for stream based request forms (see 7.9 Snapshot stream on page 104 and 7.10 Streaming requests for change on page 104).

Considering the traditional request methods, the following flow may occur:

- A Task is being run by a ControlConstruct (and associated PCs) on a controller that needs to interact with a provider system (i.e., the ControlConstruct is running on a client controller)
    - Note that this also covers OAM Job/Task
- This Task provides instructions to construct the necessary GET/POST etc.
- The ControlConstruct directs the RequestConstructor to acquire the necessary details from the ExposureContext and construct an appropriate request to a specific provider
- The RequestConstructor constructs the request and sends via the ControlPort to the provider.
- The provider receives the request via the ControlPort which directs the request to an appropriate ReceiveFilter
- The ReceiveFilter directs
    - a GET to the ExposureContext which
        - Identifies the entities to get

        - Constructs a response including those entities
        - Sends that response to the appropriate ControlPort
    - a POST/PUT/PATCH to the ExposureContext which
        - Adjusts the entities in the corresponding ConstraintDomain
            - The POST/../etc. is intention/expectation and hence a grammar/constraint format and hence a constraint form of FC etc.
        - Triggers the ControlConstruct to run a Task related to the new data
        - Sends a response to the appropriate ControlPort
    - a control message to the ControlConstruct
- The ControlPort on the client directs the response to the ReceiveFilter
    - The ReceiveFilter passes the response to the ProcessingConstruct to progress the task appropriately.

7.9 Snapshot stream

A snapshot stream provides a flow of requested one-off measurements (including basic reading of values) where the request is made through the creation of a ControlTask (e.g., realizing a measurement job). The snapshot can be loosely synchronized across multiple devices using time of day etc.

This section is to be detailed in a future release.

7.10 Streaming requests for change

In the current solution a request for change is made using a command strategy with a synchronous response to the command and which causes the initiation of a ControlTask that then coordinates the necessary activity providing asynchronous responses.

The streaming approach allows the client to stream requests to the provider. The provider is responsible for maintaining alignment with the client expectation and dealing with the requests as it has resources available. The client may compact the stream to remove requests that have not yet been acted upon and that are no longer valid. The provider will create a task related to each request and maintain a stream of progress reports on each task as for the current solution.

This section is to be detailed in a future release.

8 Future considerations

8.1 Task flow

Task flow is described in general, but further detail of task construction with examples would be of benefit.

The state transition diagram should be drawn for the TaskLifecycleState.

8.2 Clarification of use of CD, VMF and EC

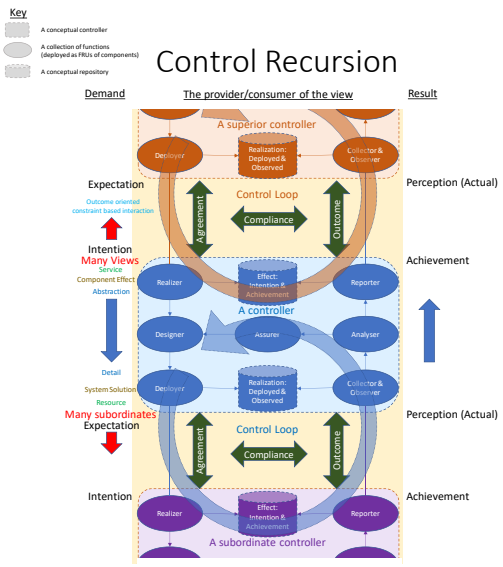
The relationship between the CD, VMF and EC need to be described further to remove ambiguities.

8.3 Control hierarchy, peering and fractals

A deeper description of the intertwining of control in terms of SDIDA loops etc. would significantly improve the applicability of this model. This should account for the fractal nature of the component-system pattern.

Controllers in hierarchy

- The diagram highlights the terminology challenges and attempts to untangle them
  - It may be beneficial to strongly define a set of sets of terms that we use rigorously (at least in places)
- Agreement/Intention/Expectation
- Outcome/Achievement/Perception
- Compliance/Discrepancy
- Demand/Results
- System Solution
- Capability
- Outcome/Experience
- Order



8.4 Client intent generation

Client has a store of realization-expectation in a constraint form which has an exposure-context that drives south bound orders via requests or streams

8.5 Intent receiver

Further details of the intent receiver will be provided in a future release of this document and will cover topics such as:

- Intent Interpreter
    - Formulating the intent
- Feeds ExposureContext with intent entities
    - Intent entities are constraint forms of FC, LTP etc.
    - Order causes a creation of a topological structure related to other intent and actual entities
    - Order is captured as an entity with associated tasks etc. (related to creation and maintenance of the outcome)
- Views:
    - Negotiation: Speculations
    - ViabilityTrial: Fragments of FC/LTP/Etc. that hold viability ranged properties
        - Fleeting, legal conflicts, potentially multiple views
    - AgreementIntention: Sparce FC/LTP/Etc. that hold constraint properties
        - Time dimension
        - Order detail (mainly persistent in a topological form (as per TAPI connectivity-service)
    - RealizationExpectation: Detailed FC/LTP/Etc. configurations (an intent structure)
        - Deployment interdependencies and ordering etc.
        - Agreement
        - Time dimension
    - ActualRealization: Fully detailed FC/LTP/Etc. with state
        - Opinion
    - IntentAchievement
        - Jeopardy
        - Compliance
        - Discrepancy

8.6 Constraint form

- Strongly related to the structure discussed during spec work

- Appears fundamental to modelling to a problem space involving functionality
- Requires the property definition to allow for a statement of ranges etc.

8.7 Forms of log

The model supports various forms of log that need to be described in more detail.

End of document