



Core Information Model (CoreModel)

TR-512.A.2 Appendix – Model Structure, Patterns and Architecture

Version 1.3.1
January 2018



ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.3.1

Disclaimer

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Any marks and brands contained herein are the property of their respective owners.

Open Networking Foundation
2275 E. Bayshore Road, Suite 103, Palo Alto, CA 94303
www.opennetworking.org

©2018 Open Networking Foundation. All rights reserved.

Open Networking Foundation, the ONF symbol, and OpenFlow are registered trademarks of the Open Networking Foundation, in the United States and/or in other countries. All other brands, products, or service names are or may be trademarks or service marks of, and are used to identify, products or services of their respective owners.

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

Disclaimer	2
Important note	2
Document History	4
1 Introduction	5
1.1 References.....	5
1.2 Definitions	5
1.3 Conventions	5
1.4 Viewing UML diagrams.....	5
1.5 Understanding the figures.....	5
1.6 Appendix Overview	5
2 Introduction to this Appendix document.....	6
3 A progression patterns – intertwining and unfolding	6
3.1 Hypergraph pattern	6
3.2 Component – System pattern	10
3.2.1 Aspects not yet covered	13
3.3 Encapsulation pattern	13
3.3.1 Related modeling principles	13
3.4 Transfer – Transform pattern	14
3.5 Realized Potential	14
3.6 Protocol "layering".....	15
3.7 Forwarding phases	16
3.8 Information architecture	17
3.9 Views of the architecture	17
3.10 Deriving relevant application models.....	18
3.11 The Component and the Classes in the CoreNetworkModel	18

List of Figures

Figure 3-1 The basic hypergraph model fragment.....	7
Figure 3-2 A constrained hypergraph	8
Figure 3-3 A constrained hypergraph	9
Figure 3-4 A recursive hypergraph	10
Figure 3-5 A recursive hypergraph	10
Figure 3-6 A recursive hypergraph	11
Figure 3-7 A generalized component.....	19

Figure 3-8 A system of components	19
Figure 3-9 A system of components viewed as a component	20
Figure 3-10 A Forwarding Component.....	20
Figure 3-11 A Termination Component.....	21
Figure 3-12 Network of Forwarding Components and Termination Components	21
Figure 3-13 Network showing Link and Trail.....	22
Figure 3-14 Forwarding.....	22
Figure 3-15 A Termination	23
Figure 3-16 LTP and LP In physical port context.....	23
Figure 3-17 Associating ports of components.....	24

Document History

Version	Date	Description of Change
		Appendix material was not published prior to Version 1.3
1.3	September 2017	Version 1.3 [Published via wiki only]
1.3.1	January 2018	Addition of text related to approval status.

1 Introduction

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

1.1 References

For a full list of references see [TR-512.1](#).

1.2 Definitions

For a full list of definition see [TR-512.1](#).

1.3 Conventions

See [TR-512.1](#) for an explanation of:

- UML conventions
- Lifecycle Stereotypes
- Diagram symbol set

1.4 Viewing UML diagrams

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

1.5 Understanding the figures

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

1.6 Appendix Overview

This document is part of the Appendix to TR-512. An overview of the Appendix is provided in [TR-512.A.1](#).

2 Introduction to this Appendix document

This document explains the model patterns and architecture that underpin the ONF CIM. The document:

- Works through the key patterns in the model such as hypergraph and component-system.
- Discusses the fundamentals of transport and how these are described in terms of the patterns
- Shows some high level examples of the patterns
- Explains how the patterns can be intertwined to form the architecture of the network from a control perspective

The descriptions in this document are built from descriptions in earlier referenced works.

3 A progression patterns – intertwining and unfolding

This section works through rationale for the formation of the model by considering the essential underlying patterns that govern the inherent structure of the problems space it is based heavily on work from TM Forum that was liaised to ONF.

A number of patterns are explored in a progression from the most fundamental through to more problem space specific. Each of the patterns is introduced in the context of the emerging model, first working towards an information architecture and then on to the ONF CIM.

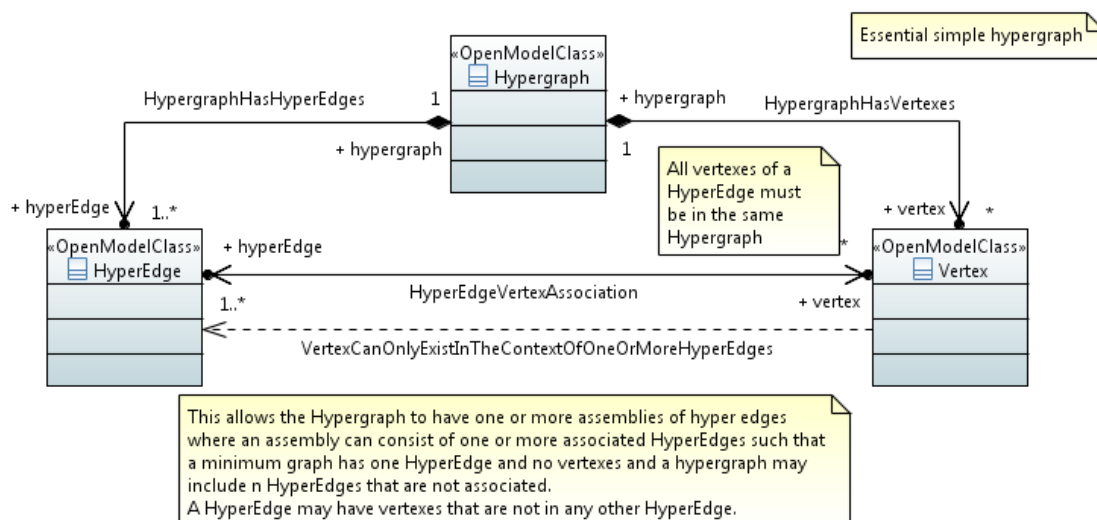
The basic underpinning of the model is essentially the representation of adjacency (connectedness) and through this adjacency there is an opportunity for some essence to pass between the adjacent places. The fundamental representation of this opportunity is the notion of a graph and due to the complexity of the problem space the hypergraph generalization of the graph has been chosen as a basis.

3.1 Hypergraph pattern

A hypergraph¹ is a generalization of a graph in which an edge can associate any number of vertices (see <http://en.wikipedia.org/wiki/Hypergraph>). The edge in a hypergraph is called a hyper edge (or hyperarc). The hyper edge can be directed (see http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382005000300005).

The basic hypergraph is represented in the UML fragment below.

¹ The hypergraph was identified as similar in pattern to the multi-pointed forwarding leading to this rationalization of the model.

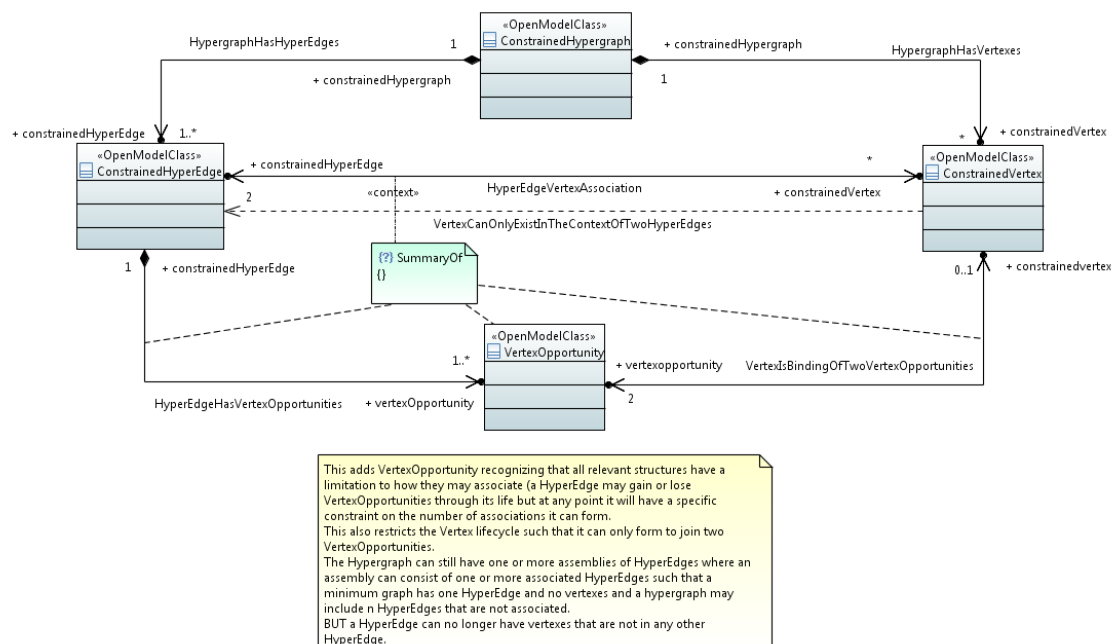


CoreModel diagram: Hypergraph-BasicStructure

Figure 3-1 The basic hypergraph model fragment

The hypergraph appears to have flexibility beyond that required by the problem space.

1. The general hypergraph allows the hyper edge to exist without any vertexes and the vertex to exist without any hyperedges. The vertex and hyper edge appear symmetric.
 - In the model above the vertex has been constrained to only be allowed in the presence of at least one hyperedge
 - The vertex appears to represent both the potential to join hyperedges and the actual joining of hyperedges
 - A constrained form of hypergraph has been developed below that restricts the vertex to only be allowed in the presence of two or more hyper edges such that the vertex represents only the joining of hyperedges. The potential to join is separated out and associated with the hyperedge as a vertex opportunity
2. A hyperedge has no restriction on the number of hyperedges it can associate with via vertexes. In the problem space there is always at least an upper bound and in many cases a specific set of opportunities.
 - The constrained form of hypergraph is defined such that joining is only allowed where there is a stated opportunity (a VertexOpportunity).
3. A hyper edge can exist with no vertexes.
 - Whilst it seems reasonable that a hyperedge may be not associated with a another hyperedge via a vertex it does not appear meaningful for a hyper edge to exist with no vertex opportunities hence the vertex opportunity is [1..*]
 - A thing with no ability to form a relationship with other things appears irrelevant. A thing is nothing without its relationship

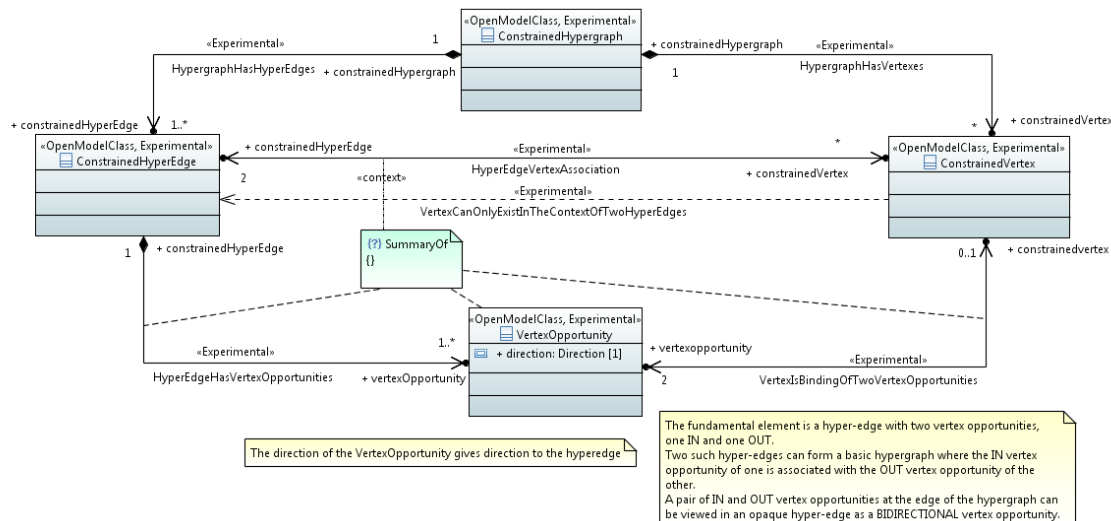


CoreModel diagram: Hypergraph-BasicConstrained

Figure 3-2 A constrained hypergraph

In the figure above the direct association between the hyperedge and the vertex is simply a summary of the associations via the vertex opportunity. The vertex only represents the joining of two hyperedges through their vertex opportunities.

The figure below adds directedness recognizing that the edge represents an adjacency only in one direction and hence there is an opportunity to pass from one vertex to another but not the reverse. The directionality has been added to the vertex opportunities rather than the edge to provide the necessary clarity.



CoreModel diagram: Hypergraph-BasicDirectedConstrained

Figure 3-3 A constrained hypergraph

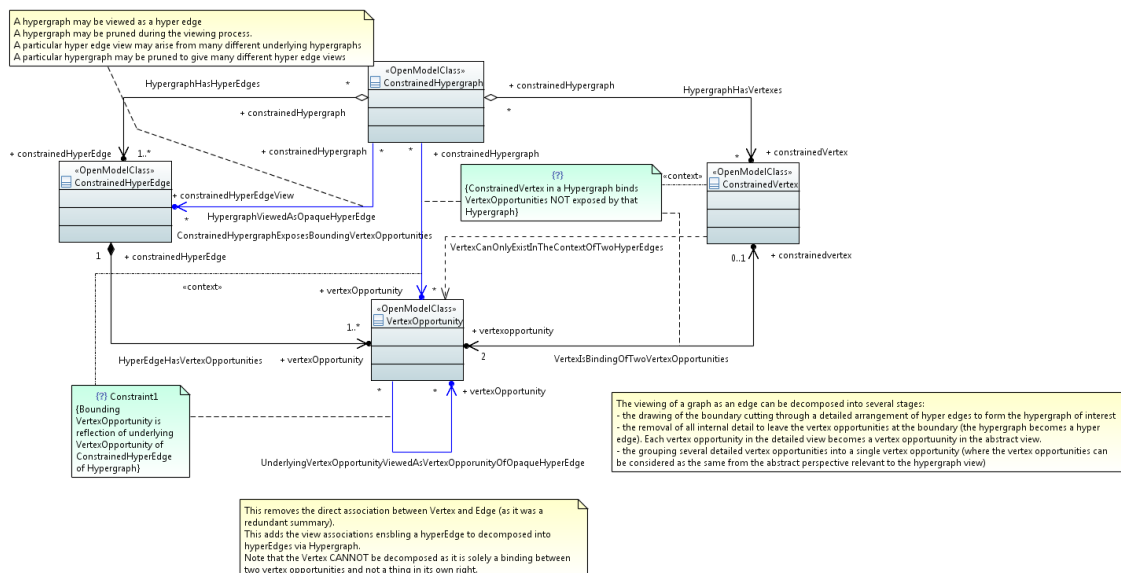
In general a directed hyperedge may represent adjacency between several ingress vertex opportunities and several egress opportunity. It is also possible that although a hyperedge groups a set of non-disjoint vertex opportunities not all ingress vertex opportunities are adjacent to all egress opportunities. This can be expressed as hypergraph encapsulated within the hyperedge where the hypergraph expresses the restrictions of the hyperedge. The association `HypergraphViewedAsOpaqueHyperEdge` in the figure below enables this.

It appears that the basic hyperedge is a unidirectional construct with a *n* ingress opportunities and *m* egress opportunities where all ingresses are adjacent to all egresses. This element appears to be the basis of all topological forms.

It is possible for a specific hyperedge that some vertex opportunities may be bidirectional and others unidirectional. This may also be expressed as an underlying hypergraph.

Some of the vertex opportunities of hyperedges in a hypergraph may be exposed as hyperedges at the boundary of the encapsulating (more abstract view) hyperedge whilst others may be internal and not exposed. The hyper edge is an opaque view of and underlying hypergraph that may be asymmetric.

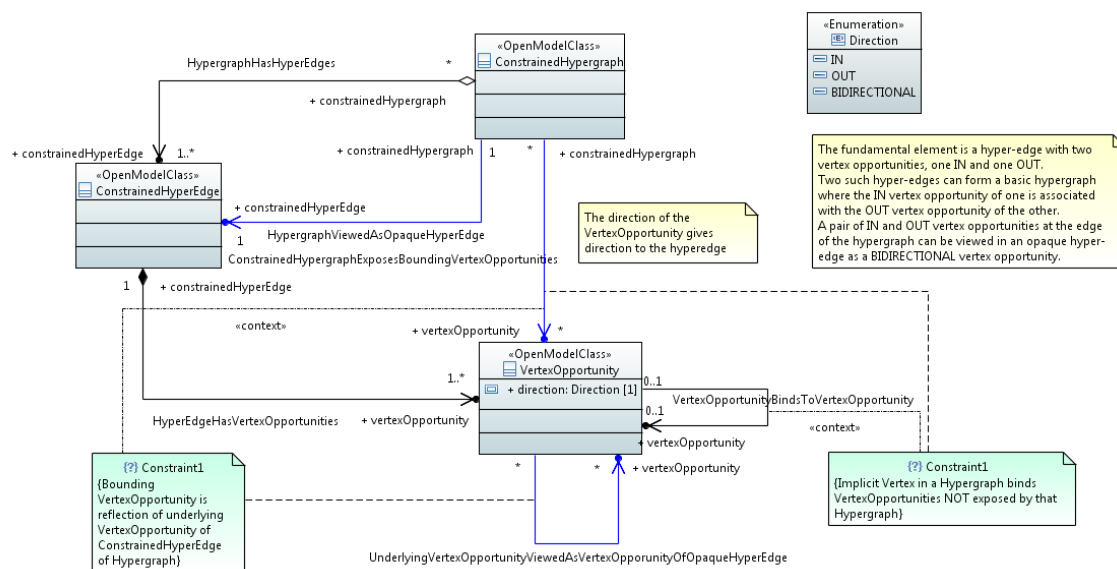
It is possible that several vertex opportunities in a detailed view can be considered as a single vertex opportunity in an abstract view. It is also possible that a vertex opportunity can appear in several abstract views.



CoreModel diagram: Hypergraph-RecursiveConstrainedForm

Figure 3-4 A recursive hypergraph

The vertex appears somewhat redundant as it now only represents the binding of two vertex opportunities and hence it can be replaced with a simple association which represents the touching of two vertex opportunities as shown in the figure below.



CoreModel diagram: Hypergraph-RecursiveConstrainedHypergraphWithImplicitVertex

Figure 3-5 A recursive hypergraph

3.2 Component – System pattern

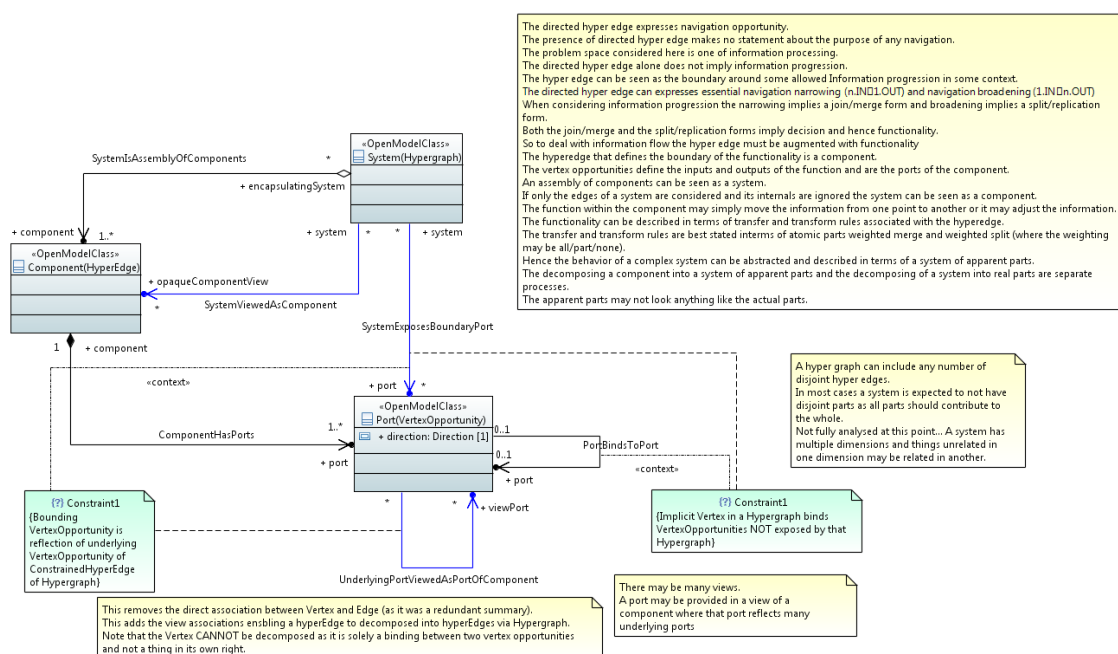
Up to this point the adjacency has been considered as invariant and absolute. In any natural context the structure will vary over time and the degree of adjacency will not be absolute. This is

considered in more detail below and in a later section. From a general perspective each adjacency can be considered as having a time dimension and variable degree of adjacency between the vertex opportunities. This variability can be expressed in terms of some adjacency function. A simple form of this could be a binary on/off consideration but in general this would be a combination of analogue properties.

As noted earlier the adjacency represents the opportunity to pass from one vertex opportunity to another. The time variable function adjusts opportunity to pass between the vertex opportunities.

Some of the variation may be "natural" whereas other variation will result from modulating control. The modulating control would essentially define the function². The modulating control is itself formed from a complex hypergraph.

As a result complex behaviour can be built up in the form of hypergraphs of time variable hyperedges. In this context the terms component and system (a complex assembly) appear more appropriate. The figure below shows the adjustment of hypergraph to relate to component-system. In this view the vertex opportunity becomes a port of the component. This is essentially the core of the component – system pattern.



CoreModel diagram: Hypergraph-RecastingRecursiveHypergraphAsComponentSystem

Figure 3-6 A recursive hypergraph

In the communication/compute/cloud problem space the pure hyperedge is insufficient as a representation of communications and processing structure and the component – system pattern appears suitable. As noted above the component aspect of the Component-System pattern is developed from the "multi-pointed" time-varying (functional) hyperedge. From a

² Essentially this leads to decomposition to a perfect analogue gate. This argument has been cut short. There is a need to introduce a storage element and a consideration of non-linear behaviour etc. It seems

communications perspective adjacency being the opportunity to pass from one vertex opportunity to another deals with the concept of flow across the hyperedge/component (this is discussed further later) and asymmetry of the flow, i.e. that the accesses via the vertex opportunities of the hyperedge are not all equal.

In essence, the component is an extension of hyperedge where the Component is a hyperedge endowed with functionality to support processing of information³ and endowed with properties at its facets (where vertexes can be) that describe inherent asymmetry and capability.

The System aspect of the Component-System pattern is developed from the hypergraph where a mesh of Components interrelated at two sided vertexes to form a system.

The Component in the pattern is an encapsulation of a System. A fabric of interconnected components can be viewed in many ways and hence there are potentially many Encapsulation views. The process of transitioning from the System to Component is not necessarily partitioning and hence the representation of transition from one view to another is not composition.

- A component is an encapsulation view of an underlying system of components
 - It is defined by a boundary with ports
 - It obscures the encapsulated system (it is essentially opaque)
 - As it is a view it cannot be assumed that the underlying system has an actual boundary where it appears to be in the view, i.e. there is no enforced containment
 - Essentially there may be many offset views of the underlying system
 - Note that the components in the system of components are also views!
- The characteristics of a component can be described in terms of an apparent system of components joining its ports
 - The apparent system of components may be quite unlike the actual encapsulated system
 - When managing a component the MCC (Management Control Continuum)⁴ component talks to the component about the components that describe its capabilities
 - An MCC component presents a view of the system of components it is managing-controlling. Another MCC component may talk to the MCC component about those components it is managing-controlling

On investigation, the traditional notion in telecoms of the fabric of a switch being the vertex (node) in a graph appeared inappropriate. The original exploration recast the Subnetwork/fabric/etc and the Link/Trail/etc essentially as hyperedges. It was noted that a vertex could be limited to one pair of hyperedges. Subsequently it became apparent that the traditional point in the telecoms model was actually also a small hyperedge and it was instead, for example, the interface formed between a facet of the hyperedge representing the point and a facet of the hyperedge representing the Subnetwork that was the two-sided vertex.

³ Again the argument has been cut short here. The hypergraph sets out adjacencies, these adjacencies are applied to cases where information is made available at one side of the adjacency for use at the other. The adjacency is not transparent and functions are applied to the information as it transitions the adjacency.

⁴ Recognizes that management is control and that traditional boundaries drawn between EMS and OSS etc are not relevant architectural boundaries. The solution should be described in terms of a recursion of control.

3.2.1 Aspects not yet covered

The following areas need to be explored further:

- Directionality
 - Add input and output to the vertex opportunities
 - Show edges encapsulating edges to give bidirectional
- Merge and split of model elements over time
- Develop notion of information flow (and information passing) and note dimensions wrt essence of information and function
- Consider disjoint sets and note complex multidimensional jointedness (where it is a common property – i.e. some mediation)

3.3 Encapsulation pattern

The encapsulation pattern focusses on the notion of enclosing the detail, such as structure, information and properties, of some assembly of things in another thing such that the detail is not visible at all, or is only visible in abstract form as properties and at ports of the enclosing thing.

In a Hypergraph, a hyperedge apparent in one view may be concealing detail of a subordinate hypergraph in another view, i.e. the concealing hyperedge can itself be decomposed into a hypergraph of hyperedges and vertices where vertices of the concealing hyperedge are abstract representation of vertices at the edges of the concealed hypergraph.

Encapsulation is a process of transition from one view to another. A single model may encompass several views, and the views are not constructed by simple processes such as partitioning although in some cases they may appear to be.

To implement the pattern as a model, there needs to be representations that allow the encapsulating class to associate to the encapsulated classes and associations in such a way as to prevent illegal encapsulations (e.g. of one lone association, associations that are not related to the encapsulated classes)⁵. The rules that govern the encapsulation should be formalized and codified in a compact language representation⁶.

Within a view, encapsulation may vary across the view and may vary for any particular instances over time depending upon specific interest and allowed visibility. A variable encapsulation approach is already used for the LTP⁷.

3.3.1 Related modeling principles

- If the positional bounds of two related concept instances are coincident for their entire lifecycle, then they may be merged into a single entity instance representing the composite concept and hence share an identifier etc.

⁵ This is not simply the composite pattern as rules relate the parts and related their “ports” to the exposed “ports” of the encapsulating entity.

⁶ UML seems lacking in this area.

⁷ Generalized variable encapsulation requires further analysis.

- If the positional bound of one concept instance is a subset of the positional bound of another concept to which it is related for its entire lifecycle and where that larger concept can be considered as a dominate definition, then the entity representing the smaller concept may be subsumed into the entity representing the larger concept and hence be identified as part of the entity for that larger concept in terms of attributes of that larger concept
- If the positional bounds of several instances of a concept are all subsets of the positional bound of another concept to which they are related for their entire lifecycle and where that larger concept can be considered as a dominate definition, then they may be subsumed into the entity via a composition relationship
- If a concept instance that bridges two other concept instances (of the same or different types) is, in the particular case, devoid of anything but identity then it may be represented simply by associations between the entities representing the two other concept instances
 - The associations may be two way navigable or one way navigable depending upon the original associations
- If a concept instance that is a leaf is devoid of anything but identity, then it may be omitted

3.4 Transfer – Transform pattern⁸

The essential purpose of a telecommunications system is to make distant things seem adjacent (it provides an apparent adjacency service). This is achieved by transfer of information, and hence the fundamental function of telecommunications is the Transfer function. From the user perspective the telecommunications system is a (rather vast) Transfer Component.

To achieve transfer, there is a need for some mechanism to encode the information in a way that allows the transfer of (the client) information and hence a need to transform the information (and eventually transform the resultant information into a signal that can be propagated in a media). So the second vital function is signal Transformation and hence the second element of this essential telecommunications pattern is Transform.

In the most abstract form, the Telecommunications network is a system of Transfer and Transform Components.

Both Transfer and Transform components follow the Component-System pattern discussed above and hence have ports. These ports are bound to form a network of components where the bindings form interfaces (or points)⁹.

3.5 Realized Potential

Transfer/Transform capacity in the telecommunications system may either be available to be configured for use (potential to enable Transfer/Transform functions) or may be allocated, configured and enabled for use in some way (enabled constrained Transfer/Transform functions)

⁸ Modelling the development of Component-System to Transfer-Transform is for further study.

⁹ It is important to distinguish between the digital layer that transfer information and the media layer that transfers signals (that may carry encoded information). It is possible to consider everything as simply information but there is a significant transition that takes place between the digital layers and the media that is best dealt with at this point by considering media as transferring signal.

or may be being used (used Transfer/Transform functions). From a Management/Control perspective the potential to enable functionality and the enabled constrained functionality are two key considerations.

The Transfer – Transform pattern can be duplicated and intertwined in such a way as to form a two level model where one level represents the potential functionality (which is essentially a pool of available capacity) and the other the enabled functionality (which is essentially capacity used from the pool).

Because of the hypergraph nature of the Transfer – Transform pattern, these two levels are interdependent hypergraphs where the topology of the enabled functionality hypergraph is necessarily the same as that of a subgraph of the potential functionality hypergraph.

Because of the Component System nature of the Transfer – Transform pattern, a view of the functional hypergraph may be taken where subgraphs of that hypergraph are encapsulated. Each subgraph is encapsulated in a single Transfer/Transform Component. The whole assembly of Components constructed via these encapsulations forms a more compact and abstracted functional hypergraph.

3.6 Protocol "layering"

First it is necessary to clarify the Transform function by considering the following

- There is some need to transfer "payload"¹⁰ (Client) information to achieve the "proximity" service noted earlier
- A protocol is invented and defined in terms of some data structures that carry information necessary to enable the transfer of the "payload" information.
- The protocol is defined to enable a transmitter to be constructed that can format a signal as per a defined structure to carry the necessary information to a receiver so that that receiver can then decode the structure and extract the "payload" information with no relevant change to that "payload" (it may be delayed a bit etc. but that is within its spec)
- The protocol includes various structures of "maintenance" information to ensure error detection and potentially error correction within the bounds of some defined operations limit, as well as to enable various other integrity validation and communication mechanisms to operate that are deemed necessary or considered useful in the transfer of the "payload" information. Some elements of the protocol may be optional
- Any receiver that can interpret all of the mandatory structure of the protocol is sufficiently conformant with the protocol to provide basic interoperability
- It is possible that two protocols are defined that have some subset of interoperability (like OC3 and STM1), and hence some protocols may be semi-compatible (as per the OC3/STM1 example).
- The protocol can be defined in terms of Characteristic Information (CI) which includes the mandatory "maintenance" information of the protocol along with the "payload"

¹⁰ This is not an ideal term. The provider of the "payload" does not necessarily pay anything directly. The key is that the information of the payload is the thing to be carried and carrying the payload is the purpose of the network and by doing so it supports and achieves the desired outcome of the proximity service. An alternative term is workload.

information container structure (but clearly not content). There will be constraints on properties of the payload information for it to be compatible with the container.

- It is possible that several streams of (client) information could be conveyed together within one payload envelope, i.e. the protocol and hence the transform process is defined to achieved as this

The flow of the signal from transmitter to receiver is itself a flow of encoded information. This information flow may be the (client) payload of another protocol "layer".

Hence the Transfer/Transform-potential/enabled structure is recursive, enabling the formation of a client-server stack of protocols (layers) that support the transfer of any particular "ultimate" ¹¹ payload where that payload may be made up of one or more client streams as may any transformation further down the stack.

It is important to note that:

- There is no specific order in the layering of protocols and, a specific protocol may appear several times in the protocol stack supporting any particular transfer of payload, the layer ordering, although not arbitrary, is in no way constrainable
- Although protocol overlap is relatively limited, signals may pass directly from one protocol domain to another so long as they are constrained to the compatible subset

Hence overemphasis on layering rigidity or protocol purity in the model may prevent the model from readily supporting all cases.

3.7 Forwarding phases

So far the model constructed from the patterns simply described:

- The transformation to a protocol to enable transmission and transfer to a receiver for transformation back to extract the payload.
- The transformation of the "protocol for transfer" into another protocol for transfer, i.e. layering
- The ability to carry several client streams in one transformed stream

The activity of transformation (the function) takes place within a processing engine of some form, but the relevant transfer discussed so far is between peer processing engines and hence "external" to those engines (ultimately over a wire, fiber or air-gap¹²).

In addition, it is often necessary to transfer a signal essentially unchanged across the processing engine¹³. Indeed the method of passing signal from one layer to another is a somewhat degenerate transfer inside a processing engine. This transfer is "internal" to the processing engine.

¹¹ Consider especially an "ultimate payload", i.e. the information that comes entirely from a source not related to construction of a telecommunications protocol such as a postal address directory).

¹² The signal transfer in these cases is at the subatomic level. This is clearly also a femto-scale system but that is not relevant here.

¹³ Obviously further rationale for this could be set out but that is not relevant to the argument here and so it is assumed to be obvious why this is necessary.

This leads to two "phases" of transfer, internal and external, where the external phase is supported by a server layer protocol and the internal phase by the processing engine.

3.8 Information architecture

The intertwining of the patterns identified in the sections above yields an information architecture. The information architecture formed can be seen as essentially the structure described in [ITU-T G.805] and [ITU-T G.800] where the two Transfer/Forwarding¹⁴ phases and the two Realized Potential phases are exposed as distinct entities:

- Subnetwork – internal potential
- Link – external potential
- Subnetwork Connection – internal enabled
- Link Connection – external enabled

3.9 Views of the architecture

The ONF CIM CoreNetworkModel provides a view of the information architecture that can be understood as a Pruned & Refactored form of the architecture that compacts some aspects of layering and that focuses on the relevant structures for Management-Control.

In the view provided by the ONF CIM the two phases "internal enabled" and "external enabled" are both represented by the ForwardingConstruct whereas the two potential phases are represented by distinct classes Link (external) and ForwardingDomain (internal).

In general, the degree of specialization depends upon the purpose and focus of the view. It is important to distinguish between entities being the same vs. being similar enough to be represented by the same abstraction. From a network architecture perspective, considering Forwarding, it appears important to distinguish between all four phases. Whereas, from an MCC perspective it seems appropriate to not distinguish in class between the internal and external phases of enabled forwarding.

Considering all viewpoints, it appears to be a viewpoint judgement as to whether everything is represented

- as a thing with properties where instances of thing only differ in properties,
- as distinct classes of thing where instances of the distinct classes only differ in properties but instances of thing differ in class and properties
- as simply distinct instances where each instance is considered as distinct and fundamentally different to any other instance

Regardless, any instance differs in at least one property from any other instance. A challenge is what is relevantly the same and what relevantly different. A further challenge is how to emphasize the sameness whilst avoiding the rigidity of classification¹⁵.

¹⁴ The term Forwarding and Transfer are synonymous in this context

¹⁵ This will be explored further in later releases.

3.10 Deriving relevant application models¹⁶

The general approach is to formally Prune & Refactor (P&R) the ONF CIM to form an application specific view, normally for use on an interface. This was the approach taken to form the TAPI model from the ONF CIM.

Whilst it is intended that in the longer term the generalized models of the ONF CIM (with expected evolution taking advantage more and more of Component-System pattern) will be used directly as the representation of the semantics of interaction, the P&R approach is taken to overcome several challenges:

- The apparent distance of the model terminology and granularity from the specific terminology of the current market place (where various different traditional terminology sets are used)
 - These different terminologies and granularities do not benefit M2M, they are there as a result of legacy H2M approaches
 - It is expected that approaches for representation of the problem semantics will continue to evolve and that the Component-System pattern will be an important factor in that evolution
- The apparent looseness of elements in the model as compared to current implementation use cases
 - There is a tendency to focus on highly specific use cases rather than patterns and to code in terminology of the specific use case.
 - The expectation is that this specificity will move to into data in a specification model¹⁷ (which could be considered as a further level of coding) that constrains the generalized model for specific cases but in such a way as to not create specialized classes and as to create a continuum
- The need for decoration of the structure with specific technology etc. detail
 - Again, using the specification model approach to decorate the generalized classes
- The need to have simple models for simple cases
 - This tends, again, to lead to specific models
 - The expectation is that a variable encapsulation approach (see section 3.3) which is related to the approach proposed in [TR-512.10](#) will be used where the model complexity is "folded-away" for the simple cases but provides the necessary sophistication for the complex cases

3.11 The Component and the Classes in the CoreNetworkModel

This section provides a sequence of figures that explain the fundamental relationship between the Component-System Pattern and the CoreNetworkModel.

¹⁶ This will be expanded in a later release.

¹⁷ See especially Scheme Spec in [TR-512.7](#).

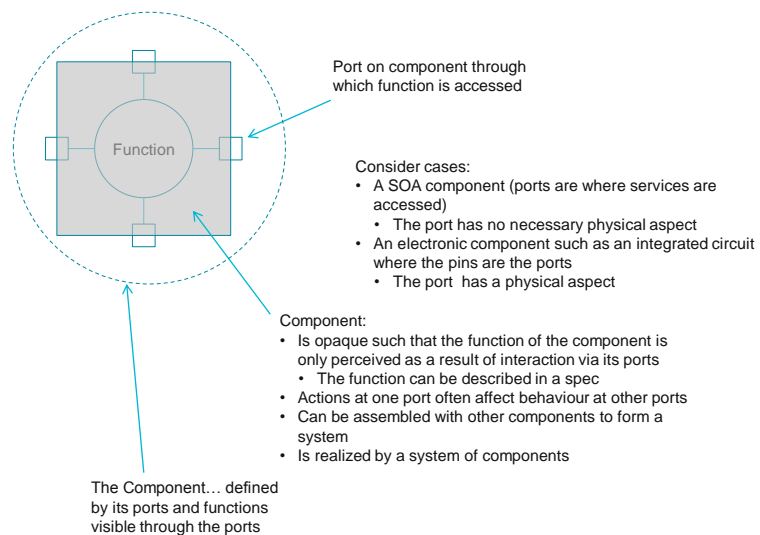


Figure 3-7 A generalized component

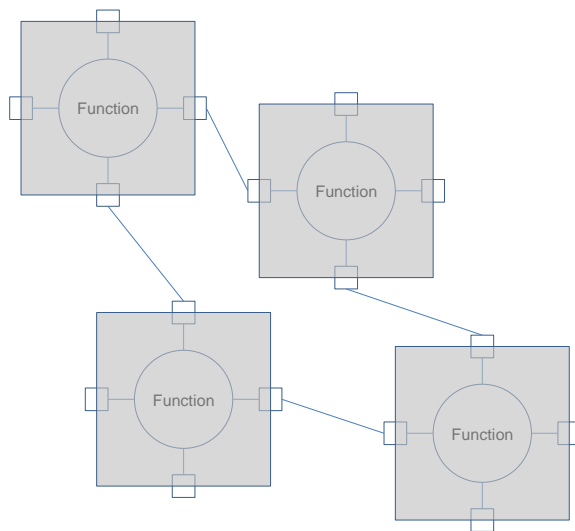


Figure 3-8 A system of components

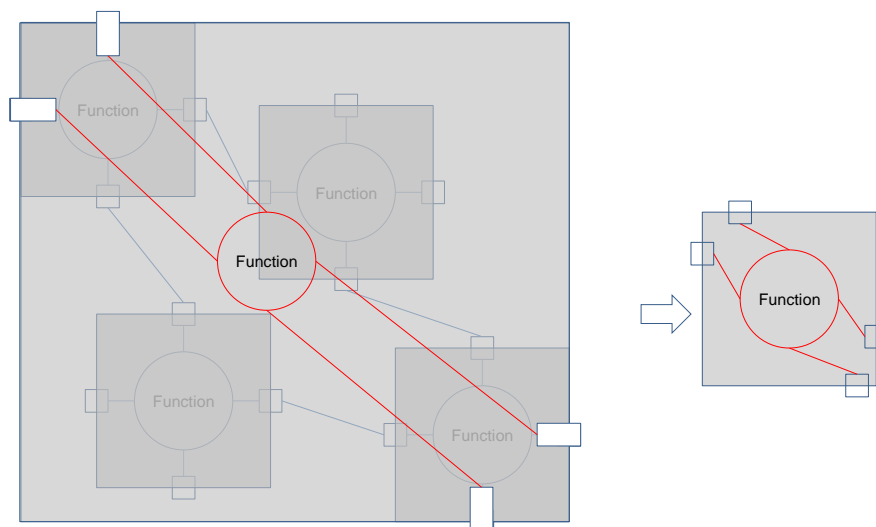


Figure 3-9 A system of components viewed as a component

- A component encapsulates components
 - The encapsulated components are in a system (this is not just a bag of components)
- The pattern is familiar as it appears in networking
 - A subnetwork encapsulates subnetworks and Links
- All functional things that can be interconnected can be considered as components
 - They expose services that are defined for the purpose of interconnecting to others

Consider the classes in the ONF CIM in the Core Network Model. Apparent adjacency is achieved by rapid transfer of information using a forwarding function. Using the component concept a forwarding function is encapsulated in a component to enable use in a system.

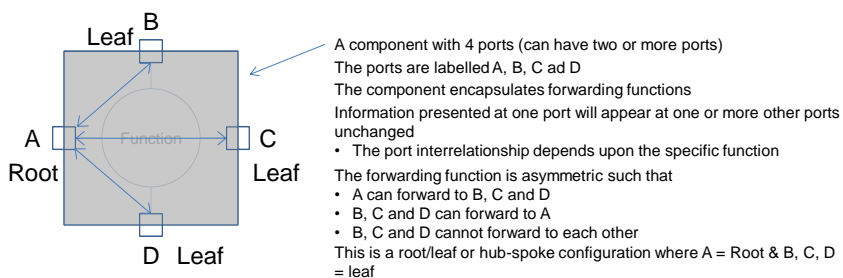


Figure 3-10 A Forwarding Component

The ForwardingConstruct is essentially a Forwarding Component.

To achieve forwarding it is necessary to carry the information over some bearer and to do this it is necessary to encode the information. This and other functions transform (and temporarily augment) the information. These transform functions are considered as Termination.

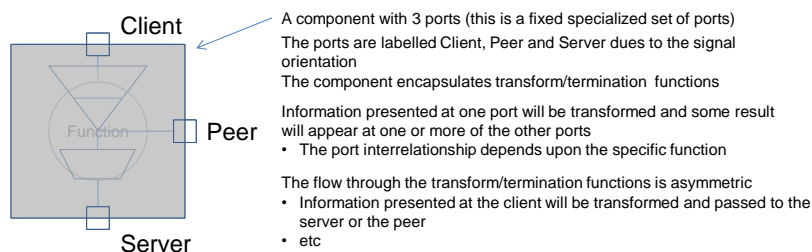


Figure 3-11 A Termination Component

The LogicalTerminationPoint is essentially a Termination Component

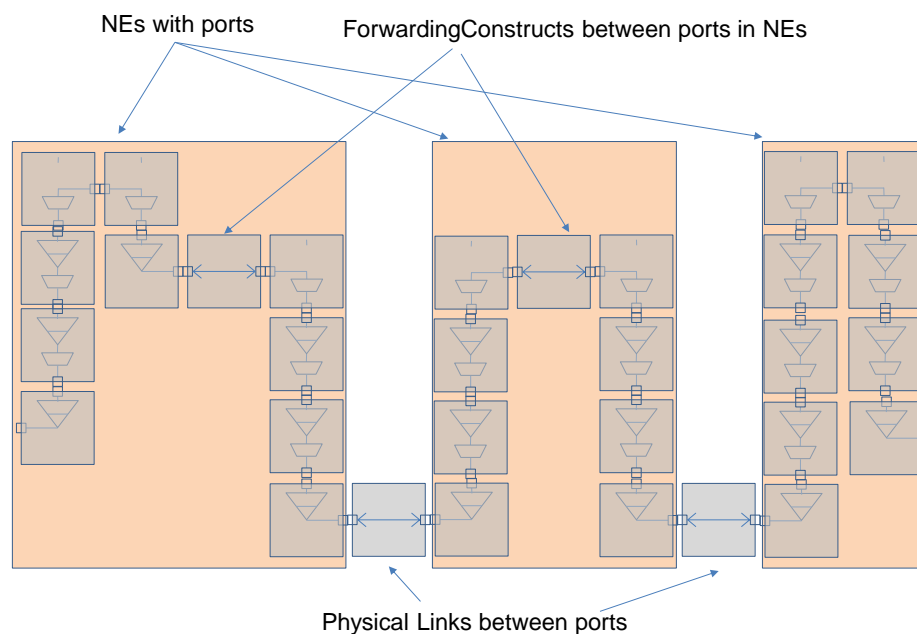


Figure 3-12 Network of Forwarding Components and Termination Components

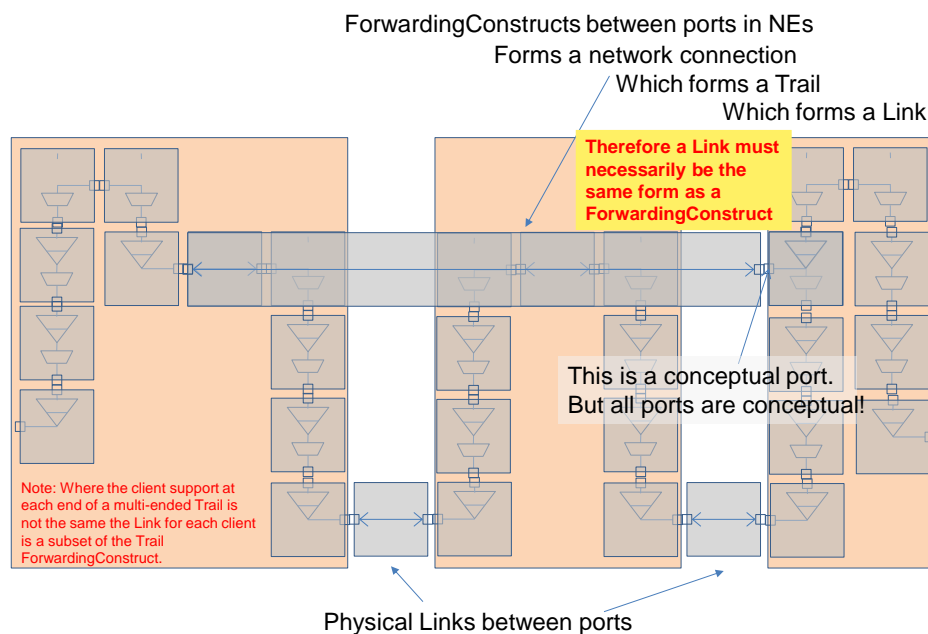


Figure 3-13 Network showing Link and Trail

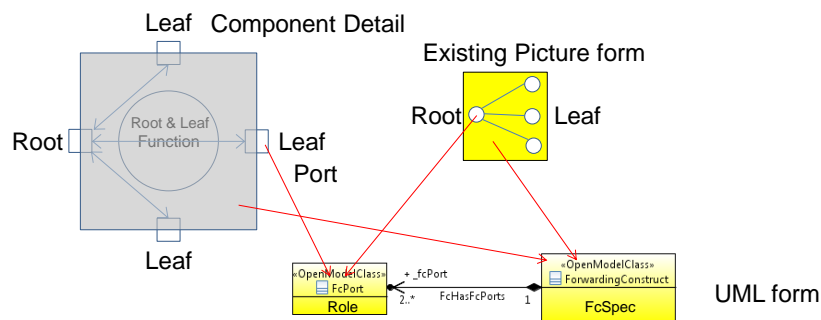


Figure 3-14 Forwarding

- Several different representations of FC
- Component ports are called endpoints in the UML model
 - Note port aligns with G.80x
- The Component has a type
- The ports (FcPorts) have roles with respect to the type

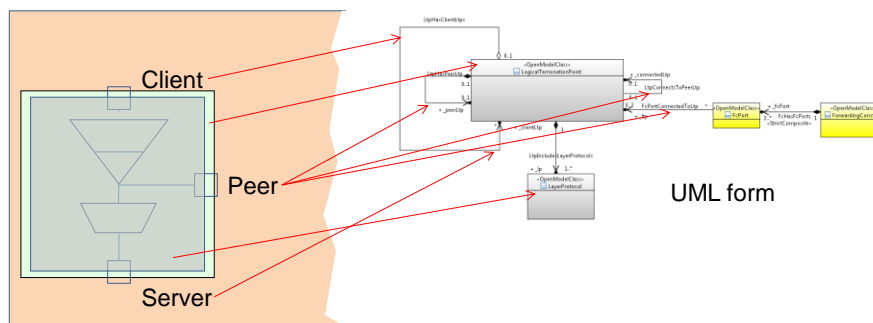


Figure 3-15 A Termination

- A list of LayerProtocol (LP) elements are encapsulated in a LogicalTerminationPoint (LTP)
- The LTP encapsulation rules are such that the LTP can expose at most one client, one peer and one server port
 - These may be from different LPs
 - The client port may give rise to multiple instances of clients and the server port may build on multiple servers (not shown in the figure above)
- The LP and LTP ports are not exposed directly in the information model but instead are absorbed in association types

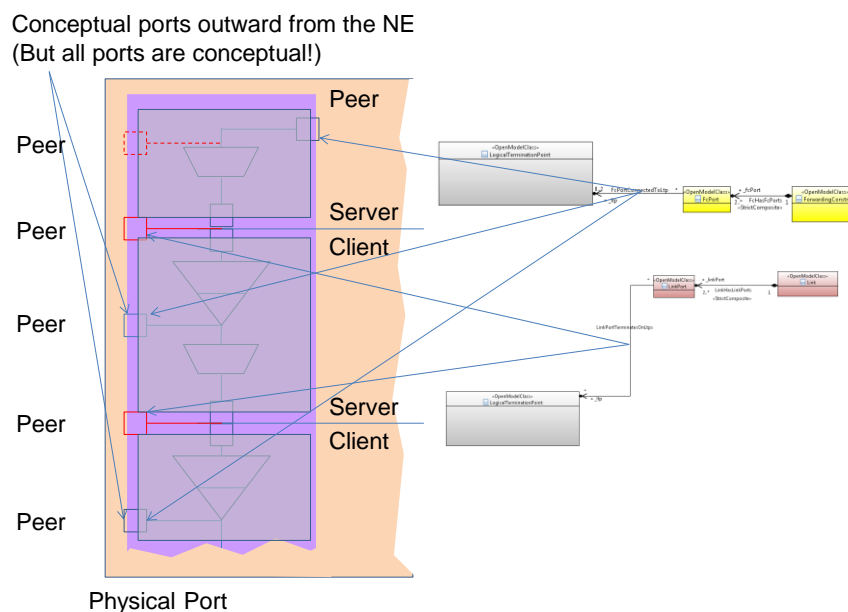


Figure 3-16 LTP and LP In physical port context

In the figure above:

- A list of LayerProtocol (LP) elements are encapsulated in a LogicalTerminationPoint (LTP)
- The LTP encapsulation rules are such that the LTP can expose at most one client, one peer and one server port
 - These may be from different LPs
 - The client port may give rise to multiple instances of clients and the server port may build on multiple servers
- The LP and LTP ports are not exposed directly in the information model but instead are absorbed in association types

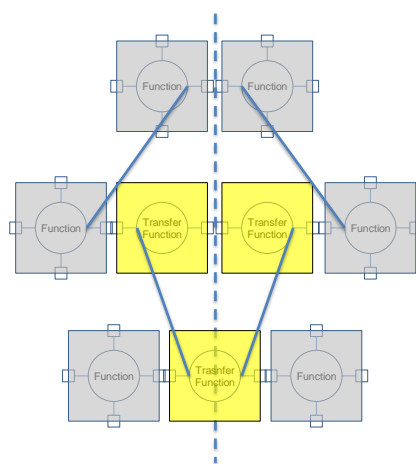


Figure 3-17 Associating ports of components

Conceptually ...

- An atomic function is at a single point
- Two functions cannot be at the same point
- Two adjacent atomic functions touch such that the adjacency is zero length
- A complex function is made by joining multiple atomic functions as a system of adjacent atomic functions
- The atomic functions forming a complex function are necessarily spread over a space
- Assuming normal space, for any particular complex function, regardless of the choice of specific arrangement of atomic functions some atomic functions that need to be adjacent to form the logic of the system cannot be actually adjacent as they are at a distant and hence must be joined via atomic functions that simply transfer information. This transfer forms a virtual adjacency.
- Adjacency between two complex functions can be represented as if a true adjacency, as for atomic functions, however in most cases the adjacency will:
 - Be achieved via atomic functions that simply transfer

- Be spread over spread over space
- Will represent multiple parallel/simultaneous semantic interactions
- Perfect transfer of information (zero time, infinite capacity) is represented in yellow in the figure as it is the essential property of a forwarding entity
- Forwarding is imperfect transfer.

End of Document