Core Information Model (CoreModel)

TR-512.7

Specification Model

Version 1.2
September 20, 2016

ONF Document Type: Technical Recommendation
ONF Document Name: Core Information Model version 1.2

Disclaimer

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

Table of Contents

List of Figures

Document History

Version	**Date**	**Description of Change**
1.0	March 30, 2015	Initial version of the base document of the "Core Information Model" fragment of the ONF Common Information Model (ONF-CIM).
1.1	November 24, 2015	Version 1.1
1.2	September 20, 2016	Version 1.2 [Note Version 1.1 was a single document whereas 1.2 is broken into a number of separate parts]

1 Introduction

This document is an addendum to the TR-512_v1.2 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 ONF Core IM - Overview.

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 Specification Model

The focus of this document is the modeling of capabilities and constraints. The document considers the modeling of patterns for the representation of capabilities and constraints and specific frameworks for representing capabilities and constraints for all entities represented in the ONF-CIM.

This document:

- Introduces the dedicated forms of specification, the primary model structure used to represent the capabilities and constraints, that feature in the current model
- Works through each form
  - FC (and Link)
  - LTP/LP
  - FD/Link
- Provides a view of work in progress on a generalized pattern for specification

The specification model relates to all other models

- Core Network Model (Termination and Forwarding) described in TR-512.2 ONF Core IM - Forwarding and Termination
- Foundation model (identifiers and naming) described in TR-512.3 ONN Core IM - Foundation
- Topology model described in TR-512.4 ONF Core IM - Topology
- Resilience model TR-512.5 ONF Core IM - Resilience
- Physical model TR-512.6 ONF Core IM - Physical

A data dictionary that sets out the details of all classes, data types and attributes is also provided (TR-512.8).

3 Purpose and essentials of the specification model

3.1 Background

The Core model classes represent fully flexible capabilities. In real deployments there are restrictions in capability due to various factors including need for low cost specific solutions. The essential approach proposed is to associate an instance of a Core model class with a set of constraints that account for the specific case.

There are several related needs that have given rise to the specification model:

- Variety: Representing the set of rules related to capabilities and restrictions for each specific case of use/application of the model
- Extension: Enabling the introduction of run time schema where the essential structure of the core model is known up front (at design/compile time) but the usage/application specific details are not known. Subsequently, the detail is added via a run-time reference to a schema that describes attributes and structure that augment the core model at runtime.
- Profile: Reducing the clutter in an instance representation where a set of details take the same values for all instances that related to a specific case (reference the case specification)[1]

[1] This relates to usage of profiles and templates etc. The specific structures used here are applied such that the structures and values in the spec are unchanging. This is not intended to be used as a common point for configuration change.

The combination of the above resulted in a separation in the model of definitions of structure (with core common content) and variable content such that instance of classes from one model fragment could point to another model fragment to enable the acquisition of that fragment of definition of the class and its subordinates at run time.

This approach is not new and many key aspects were inspired by work in the TMF SID partly described in [TMF SID 5LR].

The aim of all specification definitions is that they be rigorous definitions of specific cases of usage and enable machine interpretation where traditional interface designs would only allow human interpretation.

Whilst the mechanism allows for proprietary extensions, the intention is that primarily standard forms be used to augment the model.

3.2 Cases considered

It was recognized that the specification capability would be required by all classes but that some key per class capabilities were especially important in model. The mechanism was developed using dedicated structures for the following cases (roughly starting in the order below – although clearly much of the work was in parallel):

- Forwarding Spec
    - Main focus to provide a representation of the effective internal structure of an FC accounting for:
        - Asymmetric flow between FcPorts
        - Arrangements of switches and controllers
    - Additionally the representation of layer protocol specific attributes
    - Covers FC cases such as "Root & Leaf" and "Dual homed resilience"
        - Also fully applicable to the Link as this reflects the structure of the supporting FC
- LTP and LP spec
    - Main focus to provide a representation of
        - Layer protocol specific parameters
        - Layer protocol adaptation hierarchy rules
        - Termination flexibility rules
    - Provides a mechanism through which to acquire technology specific definition from other specification authorities
        - To also enable proprietary extensions within a technology definition
    - A critical consideration here is that ONF don't own the technology definitions:
        - New technologies need to be introducible with no change to the core
        - Proprietary extensions need to be introducible
        - Approach is to combine pruning & refactoring with the spec model
- FD and Link spec
    - Main focus on capacity and forwarding enablement restrictions

- Equipment spec[2]
    - Main focus to provide a representation of:
        - Equipping constraints
        - Functionality emergent from Equipment configuration
- Generalized spec pattern
    - Main focus to provide a common representation of
        - The mechanism for relating a class to its spec, accounting for implementation needs
        - Categories of specification element of the specification via stereotypes[3] that guide tooling and solution code
    - Note that this has been developed by refactoring the class specific specifications forms (mentioned in the bullets above) to result in a single generalized specification form.
        - The intention is that this single generalized form be used in place of the specific forms and that it is shaped by data to represent the specific structure of each of the dedicated specification forms (and any further specification forms necessary)

3.3 Resultant representations and principles

It was recognized that UML itself augmented by stereotypes provides all necessary structure and definition capability to enable representation of the specifications[4]. It was also recognized that the key is in the representation of the relationship of a class that is to be instantiated to a model definition that is not known until run time such that the instance of the instantiated class can point to the model definition (schema) for parts of its content.

The approach uses association stereotype with a member-end attribute stereotype in the class model. This attribute stereotype indicates that the member-end attribute in an instance of a class and will reference a case of a schema rather than the usual instance of a class.

A specification is used to extend the class definition BUT is constrained such that each class in the model has a particular dedicated specification structure.

The specification structure will restrict the statements that can be made and may also restrict the source of the attributes that can be applied to the specification. For example the LP specification has a place where layer protocol specific attributes can be added but it is expected and required that these attributes be derived from a formal layer protocol definition by pruning and refactoring that definition[5].

It is vital that the use of the specifications be constrained so as to enable the extension capability without allowing a model free-for-all that eventually causes the model structure to be lost.

---

[2] Note that the Equipment spec work is currently in TR-512.6 ONF Core IM - Physical. It is likely that the capability definitions for the other aspects of the model will be moved to the documents that define those aspects and this document will eventually cover the generalized specification pattern model and usage of that pattern.
[3] Currently highly experimental
[4] It appears that raw UML does not directly support any augmentation via extension schema mechanisms
[5] This will be described in detail later in the document.

3.3.1 Composition rather than inheritance

3.4 There are several UML techniques for model structuring and extension. The specification essentially uses a composition approach to extend/augment to inheritance). The composed parts are controlled by the specification attributes of the specifications will be augmented with stereotypes that direct their usage (see 4.4 Acquiring the specifications run-time

The following sketches show how a controller (or other system requiring details of the specifications) could acquire the specification on discovery of a previously unknown controllable thing.

The figures are intentionally generalized. The controllable thing could be a network element or another controller.

3.4.1 Initial system arrangement

The figure below shows a controller that has been designed to understand control of a layered network but that does not yet know any specific layer-protocols or parameters associated etc. The Controller is not yet aware of the Controllable thing, its instance repository is empty. There is a schema repository known to the Controller.

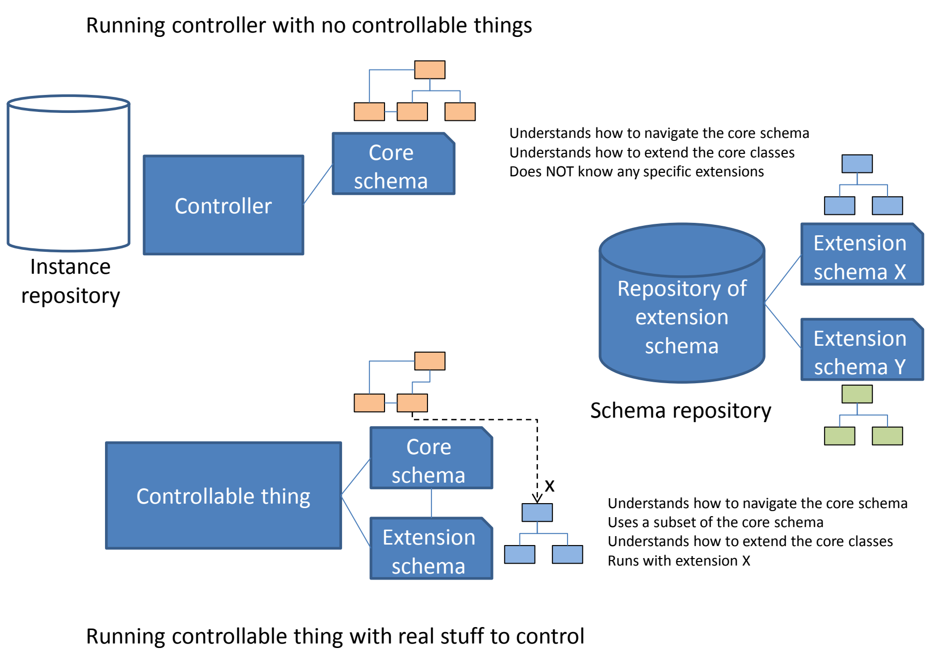


Figure 4-31 Controller and Controllable thing not yet connected

6 Some use is still made of conditional composition but it is expected that the specification approach will be used to drive all conditional content.

3.4.2 Learning to control the Controllable thing

In the following figure the Controller has been asked to control the Controllable thing. It has connected and is retrieving information.

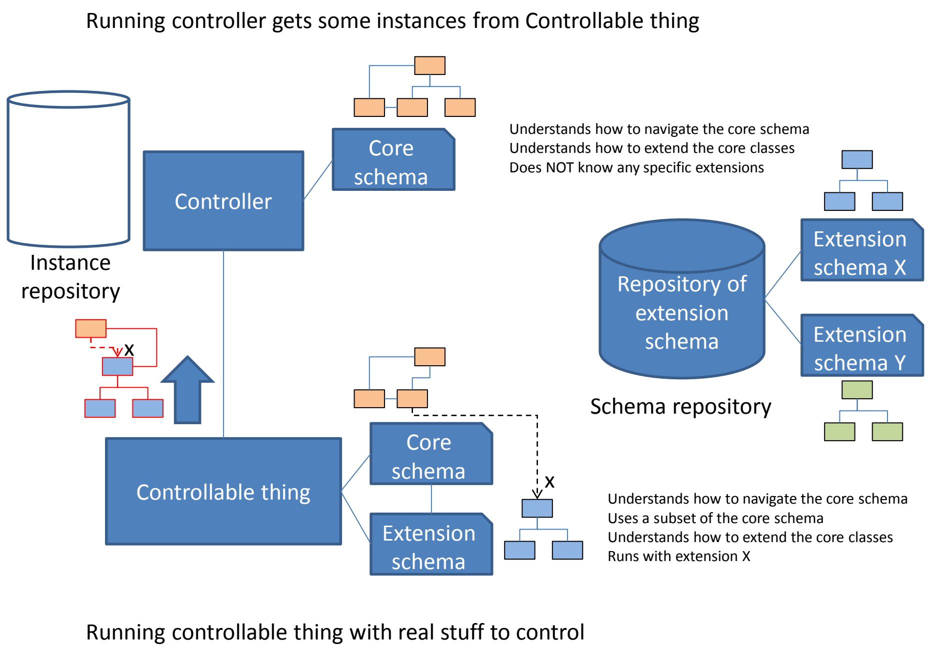


Figure 4-32 Controller connected to Controllable thing and retrieving information

The figure below shows that the Controller has stored the information it retrieved from the Controllable thing in its instance repository and has recognized that some information is not yet interpretable but that the information is related to a schema "X". The controller has asked the Schema repository if it has schema "X" and that schema is being sent to the controller. Schema "X" is a run-time version of a case of a specification (described earlier in this document).

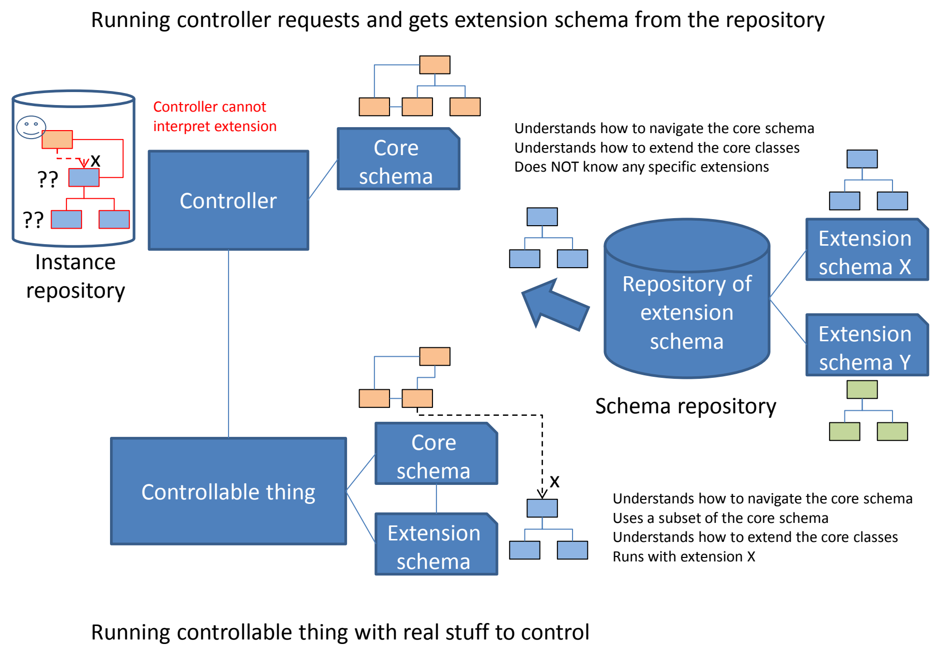


Figure 4-33 Controller acquires information on schema "X"

In the figure below the Controller now has schema "X" and hence:

- Has all invariant details for the discovered aspects of the Controllable thing (including information not known to the Controllable thing
  - These invariant details include simple information and rules
- Has the definitions of the dynamic properties
  - This includes read-only or read/write
  - Default values
  - Legal value ranges
  - Etc

Hence the Controller can interpret the attributes and in a basic solution provide the user with the opportunity to display and, as appropriate, set the attributes.

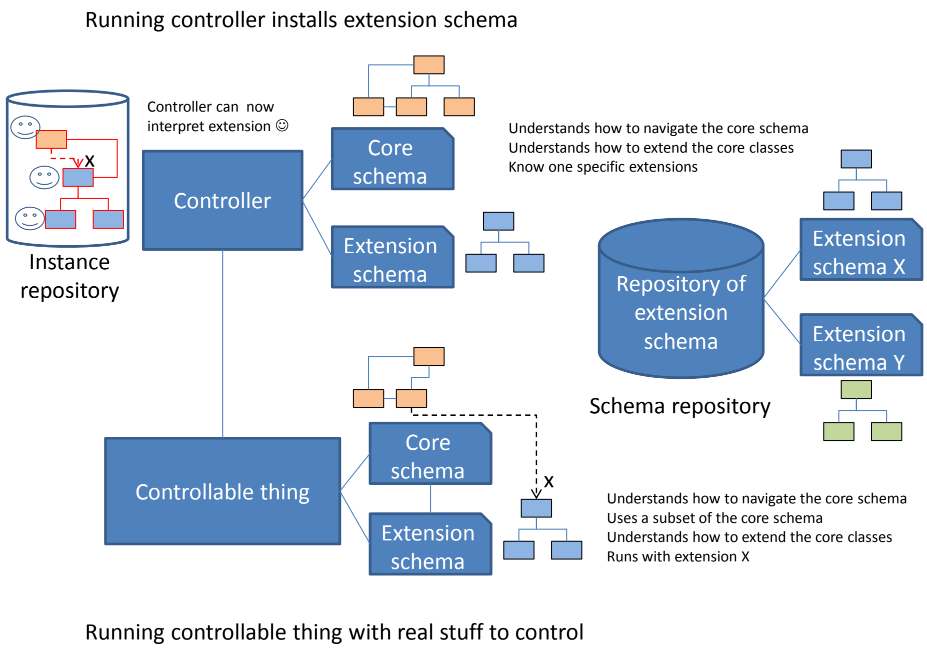


Figure 4-34 Controller can interpret and use attributes from schema "X"

3.4.3 Implications of the above

Clearly this is a very basic and simplified walkthrough and there are further system features required to fully integrate the Controllable thing.

- Controller API definition at start-up only includes the core schema
- Controller API definition is dynamically extended on arrival of a thing with an extension
- The controller may need to also acquire behaviour to deal with the new attributes etc of the extension BUT it may be able to provide significant capability without any additional behaviour

Work on the general pattern on page 48)[7, 8].

Using composition enables the specification structure to reflect and extend the structure of the class model (e.g. the LayerProtocol class gains further sub-structuring from the specification[9]). This approach also allows the specification to incorpoate augmenting content by pruning and refactoring other related external models. So for example the specific properties of a particular layer protocol can be acquired from the definitions of the specification authority of that layer protocol (e.g. see [ITU-T G.874.1].

[7] It is also intended that the specification mechanism is capable of modulating core attributes and structure. This will also be indicated via stereotypes

[8] The model entities will also provide data and stereotypes etc to drive the operations content. This will be developed at a later stage

[9] It is expected that this specific sub-structuring will become part of the LP model itself

3.4.4 Specification model restructurings during pruning & refactoring

When a specification is being composed by deriving from an external model it is likely to be necessary to refactor that model to reposition attributes to fit the ONF model structure. During that refactoring significant flattening of the model can take place. Inheritance hierarchies can (and in most cases must) be removed and [0..1] and [1] associations (primarily composition) can be folded (i.e. enable the contents of one class to transferred into the other related class).

The original structure is not lost as it can be re-acquired if beneficial[10] via the pruning and refactoring associations in the mapping model (see [ONF TR-513]).

In addition to flattening, representation transformations are likely to be required while composing a specification. For example:

- The ONF model exclusively uses a "switch and controller" approach for protection whereas some other models use a "protection group" approach. The other model will need to be refactored to the "switch and controller" form.
- The ITU-T models have a different granularity of TP modelling where the LP is modelled as several separate equal parts (TTP, CTP etc). The ONF specification model is at a finer granularity than the ITU-T model.

When acquiring a property from external definitions via pruning and refactoring it is possible to narrow the property (reduced range etc) but NOT broaden it. The narrowing must maintain the essential semantics of the property.

3.4.5 Further discussion

The term specification and template may be confusing (profile, template and specification are used inconsistently across multiple contexts and the definitions overlap with each other). From some perspectives this is essentially a templating/substituting mechanism. In a way this is "instantiating" a filled in template to form a further class structure. The "instance" of the filled in template defines the substructure of the LP/LTP and provides the definition of attributes.

This is a general principle and the pattern has been there for many years although not necessarily well formulated.

The specification method appears necessary for dynamic APIs.

There could be methods like verify to validate that the instance of LTP supports the specification definition.

Specifications will be constructed by the designer/developer of the item being specified and will be available to the controllers via some interface. The hosting entity that provides the specification may be:

- The same as the one providing the interface that uses the specification
- The organization of the designer/developer of the item being specified
- Some central repository/library

[10] Much structuring in the models examined was bundling of attributes rather than dependency graph or flow semantic based structuring. This does not allow for enhanced model interpretation over and above that that can be achieved from the attribute alone. Work is progressing slowly on dependency graph (and flow semantic) modelling.

3.5 Adoption and migration considerations

This section briefly discusses migration from traditional design time fixed solutions to a fully run time dynamic solution using specifications. The steps discussed are examples; there are many ways of approaching this.

3.5.1 Using as a traditional interface

As noted the specification provides a dynamic run time definition for use in a dynamic API context. However, it is quite possible to use the same APIs in a more traditional static mode.

In a traditional system the entire schema has been coded prior to compile. The specification reference can be ignored run time and instead the content of the specification can be compiled into the systems on both sides of the interface.

In a slightly more sophisticated solution the specification reference can be used to validate the version compatibility of the interface.

Alternatively, assuming a JSON or XML encoded interface, the additional attributes could be transformed into simple name and value lists much like a traditional semi-dynamic API with soft properties.

3.5.2 Using with basic specs and prior knowledge

The specification rather than being fully interpreted run time could be compiled but decoupled allowing for some dynamic behavior but not for previously unknown specs to be used.

In this case the specification reference would allow selection of previously compiled schema and associated behavior.

3.5.3 Using with full specs with discovery

The spec pointer would be used to reference the library to acquire the necessary specification that would then be used to interpret the attributes that were not defined in the compile time schema.

The definition of the attributes will provide information on value ranges, defaults, writeability etc. From this the application can determine whether to make available on a configuration UI or as part of a flexible profile etc.

Clearly more sophisticated usage will require special code but on arrival of the specification this code can potentially be identified, installed and be available to run on arrival of the first case of use of the specification.

Etc…

3.6 Enabling innovation whilst removing unnecessary variety

As discussed the specification mechanism allows the raw model to be augmented at run time. As noted the augmentation may be proprietary. The intention is that the source of the augmentation be identified. The intention is also that the extension is in a place allowed by the specification authority of the area being augmented (e.g. ITU-T for protocol definitions).

There is a challenge here of hitting the right degree of allowed augmentation so that innovation is in no way stifled whilst model chaos is prevented. This will be a journey and learning experience.

4 Dedicated specification structures

It should be noted that much of this work is experimental.

4.1 ForwardingConstruct Specification

4.1.1 Overview of the Forwarding Spec

Prior to embarking on a brief description of the Forwarding Spec and associated classes it is important to explain the Forwarding Spec in the context of the specification approach in general. In this model the specification classes provide a mechanism to express the restrictions of a particular case of application of a specific class or set of classes. For example an FC in general has[2..*] FcPorts while a specific case of FC may have exactly 4. This case may also be such that it has 2 internal switches and such that these switches affect specific flows in the FC. The ForwardingSpec is designed to allow the expression of cases of this sort.

The number of FcPorts alone does not adequately characterize the FC. The figure below shows two FCs both with exactly four FcPorts. The flows through the two FCs are clearly very different. Simply knowing the labels for the roles of the FcPorts does not provide sufficient information on the flows within the FC. To fully describe the FC behavior it is necessary to represent the flows.

In effect what we are doing is to model the internals of the FC, to create a 'prototypical instance' that can then be shared by any FC or Link instances.

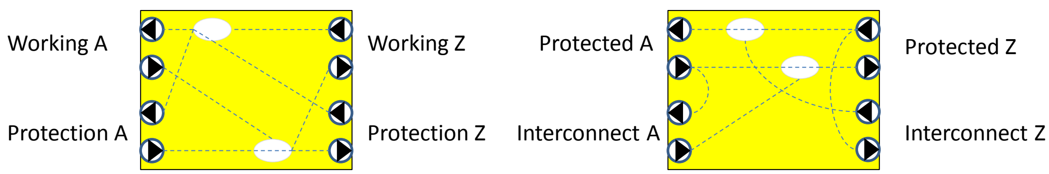


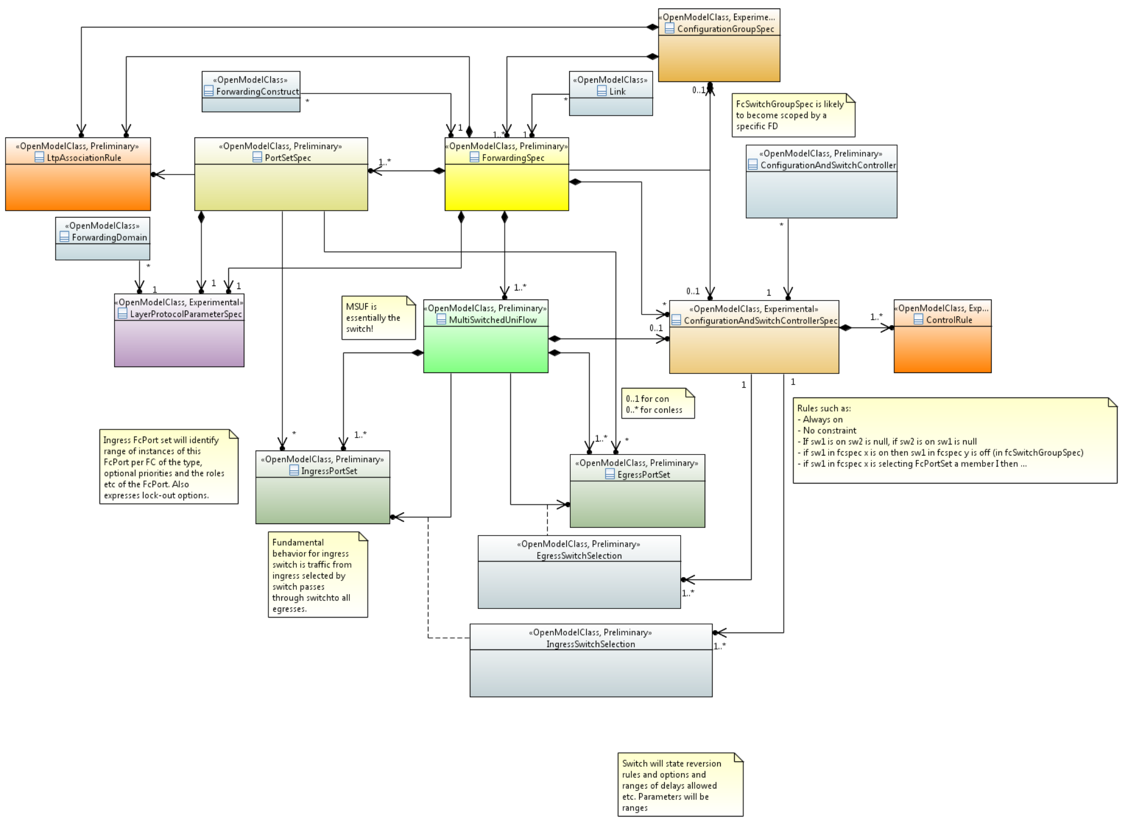
Figure 4-1 Two FCs with four bidirectional FcPorts

In the diagram below, the ForwardingSpec and supporting PortSetSpec describe the capabilities of the FC in terms of MultiSwitchedUniFlows, each of which has [1..*] IngressPortSets and [1..*] EgressPortSets. Each MultiSwitchedUniFlow may have [0..1] ingress switches and [0..1] egress switches where the ingress switch may select only one set member from one set and the egress switch may select [1..*] set members from the egress set. The ingress and egress switch selections are controlled by the ConfigurationAndSwitchController (see TR-512.5 ONF Core IM - Resilience) that may be:

- embedded in the switch when there is no coordination of switches required
- embedded in the FC when the coordination of switches is only in the scope of the FC
- independent of the FC and described by the ConfigurationGroupSpec where there is multi-FC coordination required

The behavior of the ConfigurationAndSwitchController is described by the ConfigurationAndSwitchControlSpec and associated ControlRules.

The model has been exercised for a number of different cases (some provided later in this section). The figure below provides the class diagram of the Forwarding Spec fragment.



CoreModel diagram: Spec-FcCapabilitySpec

Figure 4-2 Class Diagram of the Spec Model of the FC and FcPort

Also shown in the figure above are Link and ForwardingDomain, these will be discussed in a later section. The LayerProtocolParameterSpec is rudimentary at this point. It essentially provides a vehicle to convey layer protocol specific parameters to the FC (and Link/FD). There is a complexity here that has not been fully developed in that the parameters are invariable abstractions of properties of the LTP/LP. This will be developed in more detail in the next release.

The diagrams below[11] show a pictorial view of some of the classes above (the colors used in the figure are consistent with those used in the model above).

[11] Note that the Switched Unidirectional Flow is essentially a degenerate FC. Hence there is an opportunity to converge the FC and FcSpec models.

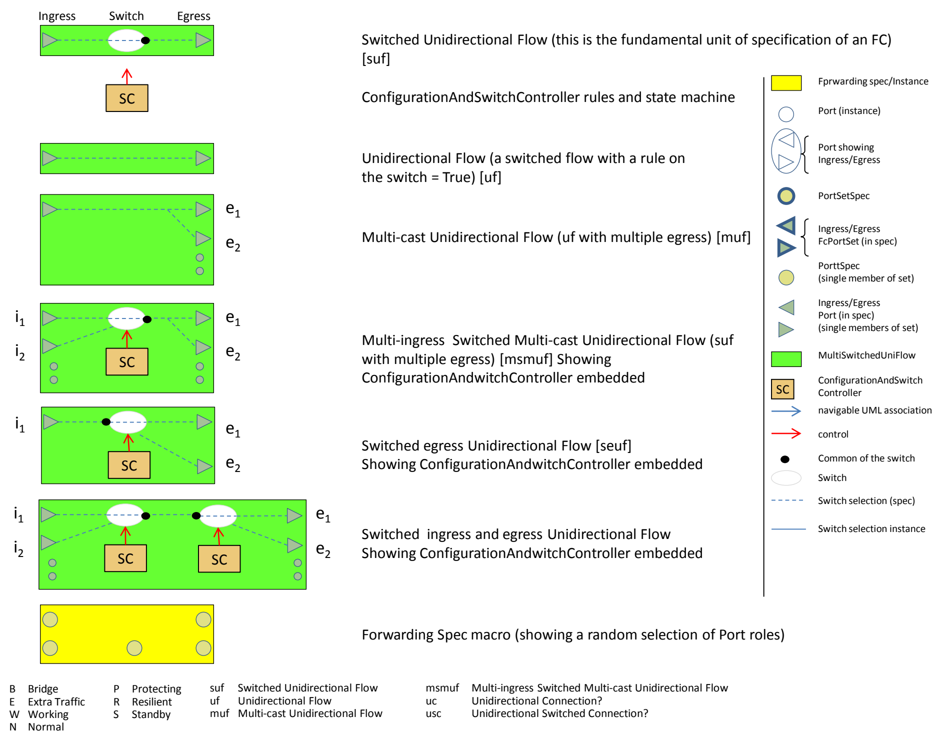


Figure 4-3 Pictorial view of the Spec Model of Configuration Control

4.1.2 Class details

4.1.2.1 ConfigurationAndSwitchControllerSpec

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::ConfigurationAndSwitchControllerSpec

The spec of a ConfigurationAndSwitchController.

Inherits properties from:

- GlobalClass

This class is Experimental.

4.1.2.2 ConfigurationGroupSpec

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::ConfigurationGroupSpec

The specification of the grouping rules for a particular configuration of FCs and CASCs.

Inherits properties from:

- GlobalClass

This class is Experimental.

4.1.2.3 ControlRule

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::ControlRule

A rule the describes the bounds of the behavior of a CASC.

Inherits properties from:

- LocalClass

This class is Experimental.

4.1.2.4 EgressPortSet

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::EgressPortSet

The grouping of FC egress ports that have the same behavior and relationship to the switch etc. Will carry rules for the grouping.

Inherits properties from:

- LocalClass

This class is Preliminary.

4.1.2.5 EgressSwitchSelection

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::EgressSwitchSelection

Rues for the control of the state of the egress switch.

This class is Preliminary.

4.1.2.6 ForwardingSpec

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::ForwardingSpec

The overall spec for the forwarding entity.

Inherits properties from:

- GlobalClass

This class is Preliminary.

4.1.2.7 IngressPortSet

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::IngressPortSet

The grouping of FC ingress ports that have the same behavior and relationship to the switch etc. Will carry rules for the grouping.

Inherits properties from:

- LocalClass

This class is Preliminary.

4.1.2.8 IngressSwitchSelection

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::IngressSwitchSelection

Rues for the control of the state of the ingress switch.

This class is Preliminary.

4.1.2.9 LayerProtocolParameterSpec

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::LayerProtocolParameterSpec

Offers the opportunity to define a list layer-protocol related parameters. Used to specify the extension a class.

This class is Experimental.

4.1.2.10 LtpAssociationRule

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::LtpAssociationRule

Rules for the association from the port spec to LTPs identifying restrictions of use.

Inherits properties from:

- LocalClass

This class is Preliminary.

4.1.2.11 MultiSwitchedUniFlow

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::MultiSwitchedUniFlow

A switched unidirection forwarding element that can take one or more inputs and switch to one or more outputs. The switch can also be open (high impedance)

Inherits properties from:

- LocalClass

This class is Preliminary.

4.1.2.12 PortSetSpec

Qualified Name:
CoreModel::CoreSpecificationModel::FcCapability::ObjectClasses::PortSetSpec

The specifciation a set of equivalent port of the forwarding entity.

Inherits properties from:

- LocalClass

This class is Preliminary.

4.1.3 Use of the Forwarding Spec

The figure below shows a pictorial view of a case of ForwardingSpec. The lower element of the diagram shows specification class instances and the upper element shows an instance of FC abiding by the spec.

The MultiSwitchUniFlow elements represent the allowed flows across the FC and explain what each switch (in this case there is one switch only) related to the FC does.

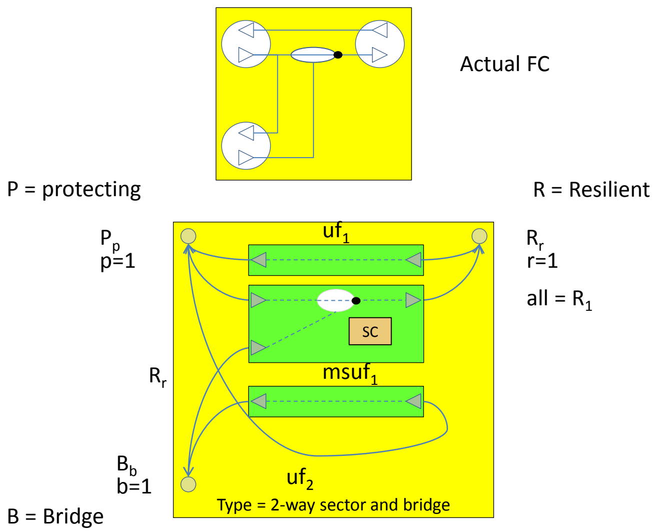


Figure 4-4 Pictorial view of spec model and resulting FC instance

The example above is a relatively complex switch (used in ladder protection schemes). A more straight forward case is shown below. It should be noted that the multiple ports on the left side of the actual FC are represented by one statement in the spec. The spec can provide a compact statement of the capability where there is a systematic structure.

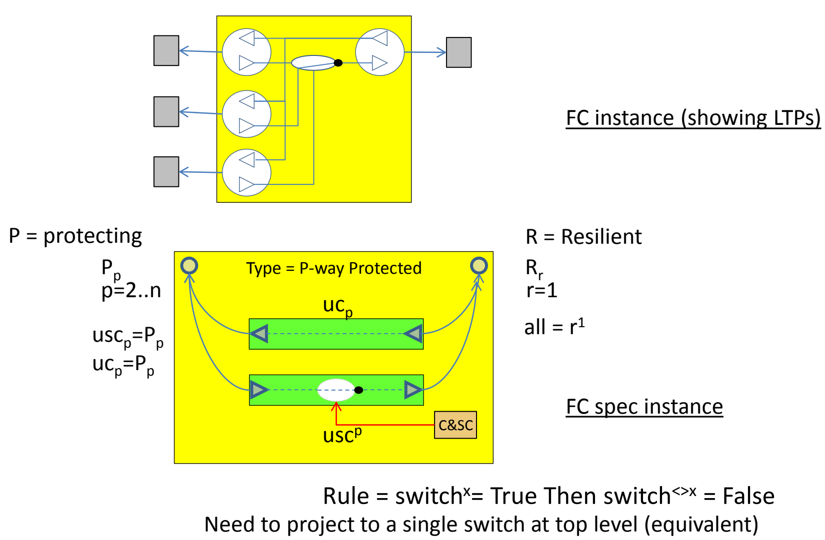


Figure 4-5 Compacting the Forwarding Spec rules

Where there is significant symmetry a very compact form can be developed. In the figure below the FC and switch arrangement is highly symmetric. It is assumed that the switches do not offer any reversion (i.e. the switch will stay where it is until there is a failure on the signal it is receiving when it will then switch to the other port. Both sides of the FC behave in the same way.

The specification forms set out at the bottom of the figure all represent the FC at the top. The one on the left is very specific and verbose and the one on the right is most compact (and assumes that there is normal case of "no loopback allowed" which forces there to be at least two ports). Clearly there is a need to state the bounds on the number ports in the PortSet which in this case, assuming the FC figure is precisely what is supported, is precisely 4 ports in two groups of two.

Note that the very compact forms are not recommended other than for unprotected packet cases as the method of constraining ports in the PortSetSpec has not been fully developed. Clearly it is important that a standard form of specification of constraints emerges so as to prevent unnecessary decoding complexity. Without a well-defined constraint the compact spec on the left would allow all sorts of FCs to be created from a two port switched unidirectional FC upwards.

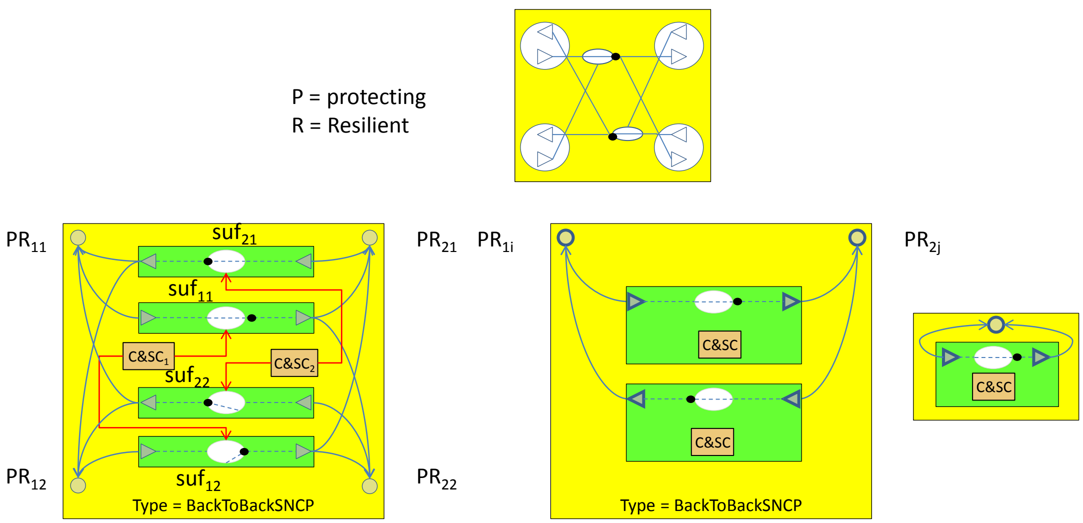


Figure 4-6 Highly compact form for symmetric FC

The figure below sets out some Forwarding Specs overlaid where the FCs corresponding to the specs would be in a 1:N protection scheme (see TR-512.5 ONF Core IM - Resilience where the spec layout represents the NE on the right in the figure in TR-512.5). The upper two FCs have the same spec. The lower FC has a different spec. The figure does not show the full arrangement of C&SCs (which would align with the protection scheme definition)[12].

[12] It has been recognized that there should be scheme specifications to deal with protection constraints on protection structures and that these should use the techniques set out in this document. This is for future development.

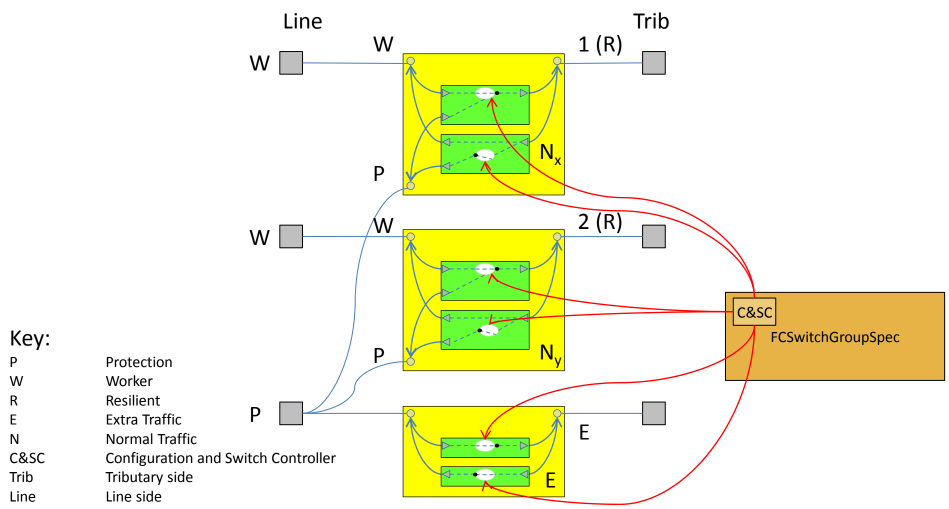


Figure 4-7 Forwarding Spec view for 1:N protection

4.1.4 Dealing with exceptional behavior on failure

The figure below shows the symmetric FC discussed earlier. Both the representation of the FC and the spec view on the left provide an abstraction that may be relevant to offer to a service oriented client. However the underlying implementation may allow a variety of undesired behaviors under various failure cases.

The spec view on the right (one direction shown only) provides a more accurate description of the opportunities for asymmetric flow but this may be considered too complex to present to the service oriented user.

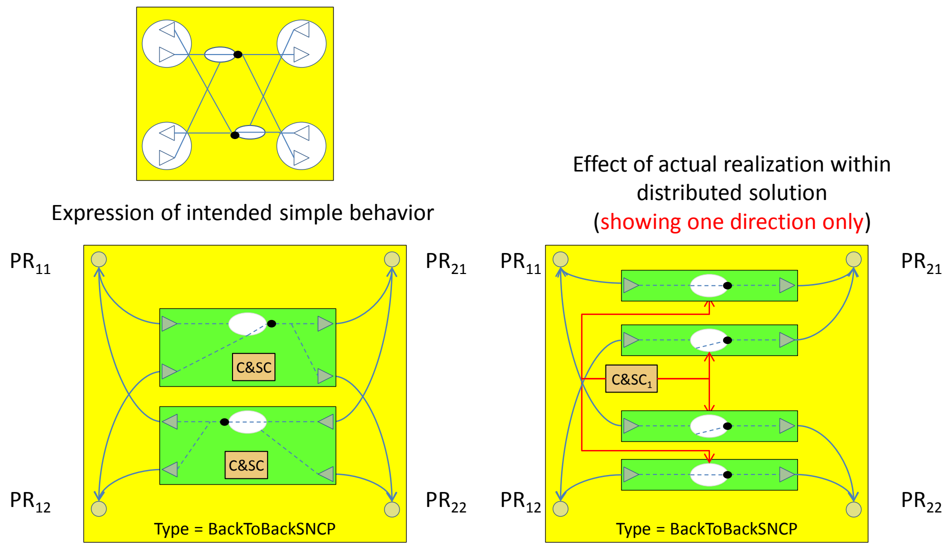


Figure 4-8 Desired abstraction and actual potential

Consider the network depicted in the figure below. This network will behave as described in the spec on the left in the figure above for all single failure mechanisms (where any egress port can be reported as failed in addition to the switch state reports). Hence under normal circumstances the spec view on the left is all that is required and is how the operator may choose to model the case.

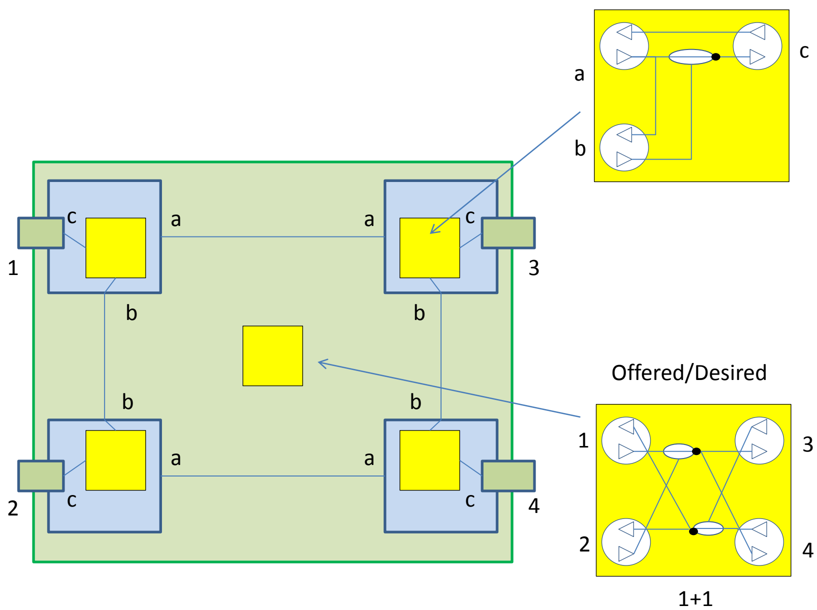


Figure 4-9 Network view of protection scheme with simplified abstract view

But how then should the operator present obscure failure cases on the rare occasions that they do occur? The figure below sets out some of the possible flow states of the configuration. The flow states shown in the red ellipse are of particular concern as they do not have the same external behavior as the claim in the spec. For example, taking the top flow layout in the red ellipse, if the client disconnects traffic from PR_{11}, traffic at PR_{21} will vanish, but PR_{22} will continue to provide traffic. Likewise if PR_{12} is disconnected, a corresponding unexpected flow will occur.

Clearly the operator may want to temporarily express to the client what the actual current state of flow is. This will be especially relevant if the client wants to perform some engineering works on their local network that may involve disconnecting traffic from one or more of the ports.

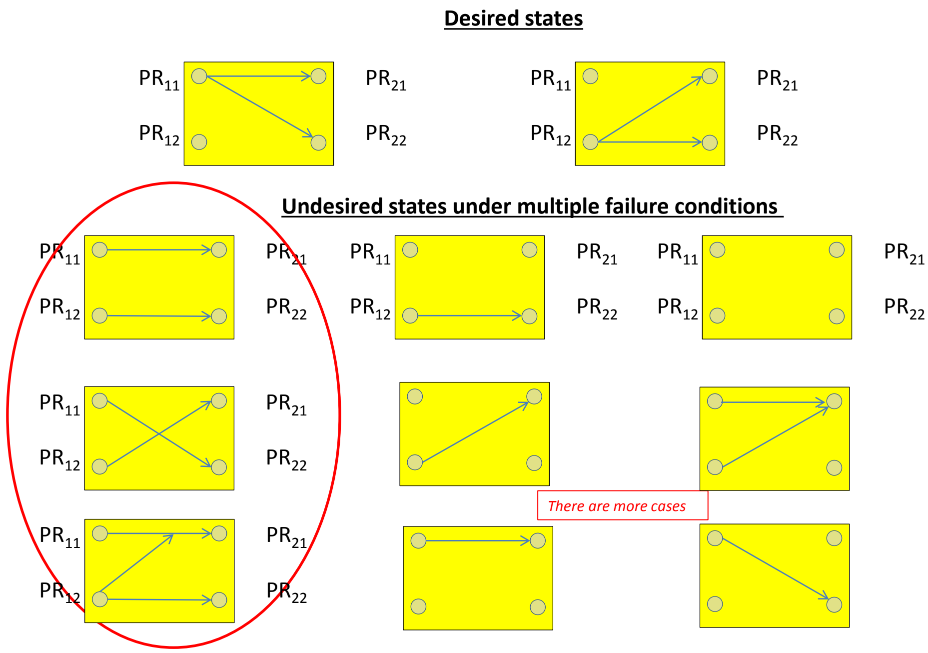


Figure 4-10 Various flow states under failure

To account for the issue highlighted in the figure above, an actual possible flow statement could be provided (along perhaps with an alert of non-normal internal flow state). The actual possible flow would be described in one or more specs that lay out the switched flows of the current snapshot of disjoint structures. This could use the normal spec structure.

The failure in the network shown in the figure below would result in the top undesired flow state in the red ellipse in the figure above.

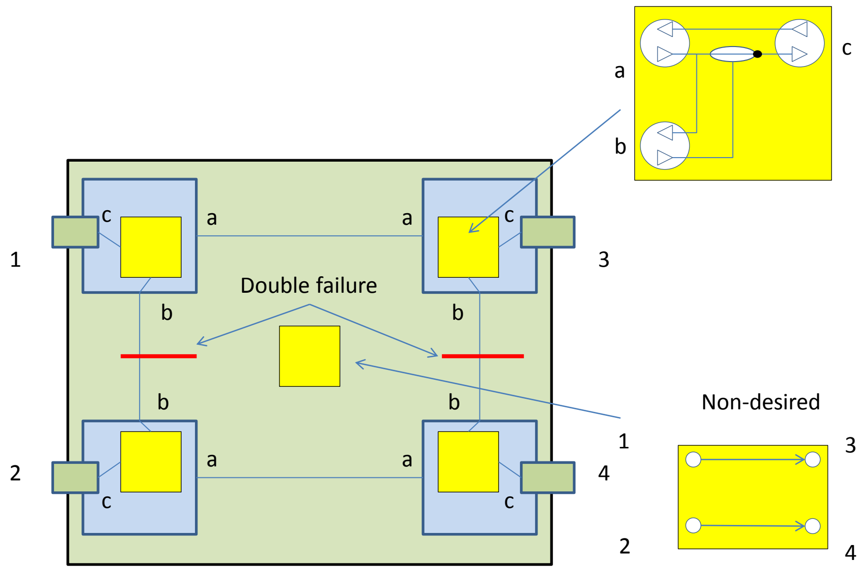


Figure 4-11 Double failure resulting in undesired flow

The figure below shows a spec instance representing the undesired flow.

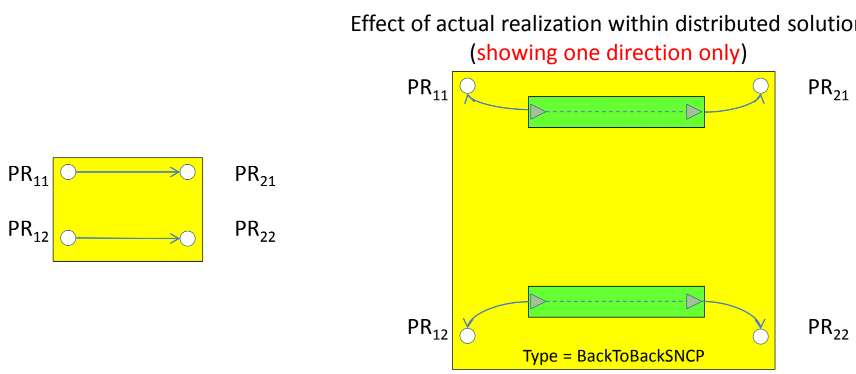


Figure 4-12 Using the Forwarding Spec model to show undesired flow

4.1.5 Spec v Instance

The spec provides all static details and identifies dynamic aspects. As the spec is available runtime the instance of the class need only identify the state of the dynamic aspects[13].

[13] As the static aspects are carried by the spec, representation in each instance of the class would be unnecessary replication.

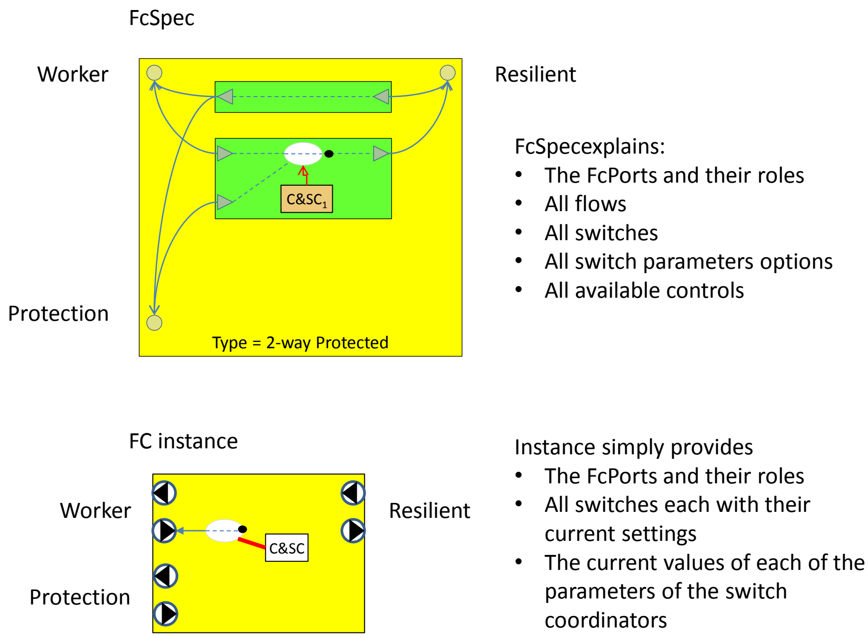


Figure 4-13 ForwardingSpec and FC instance details

The figure below shows, at the top, an example of a 1+1 protection network layout and highlights, in detail, the nodal FC (in the red dashed circle) for the right most NE (these diagrams were derived from TR-512.5 ONF Core IM - Resilience). The figure also shows a view of the FC model and a view of ForwardingSpec model. These two models are related to diagram of the instance of the FC model representing the FC in the red dashed circle and the spec that would describe the FC highlighted in the red dashed circle.

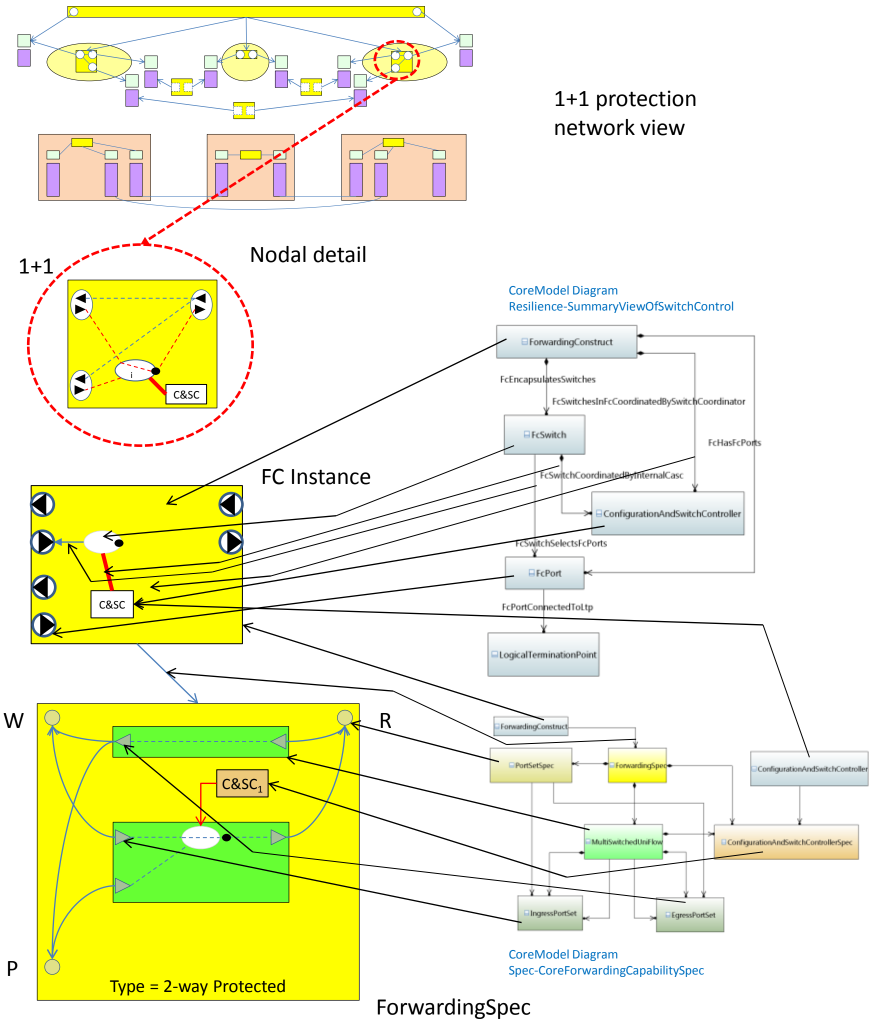


Figure 4-14 View of ForwardingSpec and FC model in the context of a 1+1 protection case

4.1.6 Role names and Port numbers

The spec also provides role Port role names and Port numbering rules.

- Where there are compact symmetric definitions the spec should provide naming rules for each dimension of unfolding of the spec (e.g. PR_{11}, PR_{21}, PR_{22} and PR_{12}).
- Role names relate to the flow and could be such Root/Leaf, Active/Standby, Worker/Protection, Balanced, Symmetric etc

4.1.7 Mapping to Open Flow

As the Forwarding Spec model is designed to break the FC down into component flows, the Forwarding Spec provides a bridge to the flow aspect of the OpenFlow[14]. Some work was carried

[14] Whilst OpenFlow can provide flow only rules it tends to be used in what is a hybrid mode from the IM perspective where definition of flow and termination is provided in single rule statements. The FC spec model only provides part of the mapping and the LTP spec model needs to be used in conjunction with the FC spec model to provide a full definition.

out as a proof of concept. This work has not been extended recently and is now somewhat dated in detail, but still provides a view of an appropriate route for mapping.

The figure below provides a sketch of the mapping between a pictorial view of ForwardingSpec statements and an older form of OpenFlow Table Type Patterns for a Root and Leaf form.

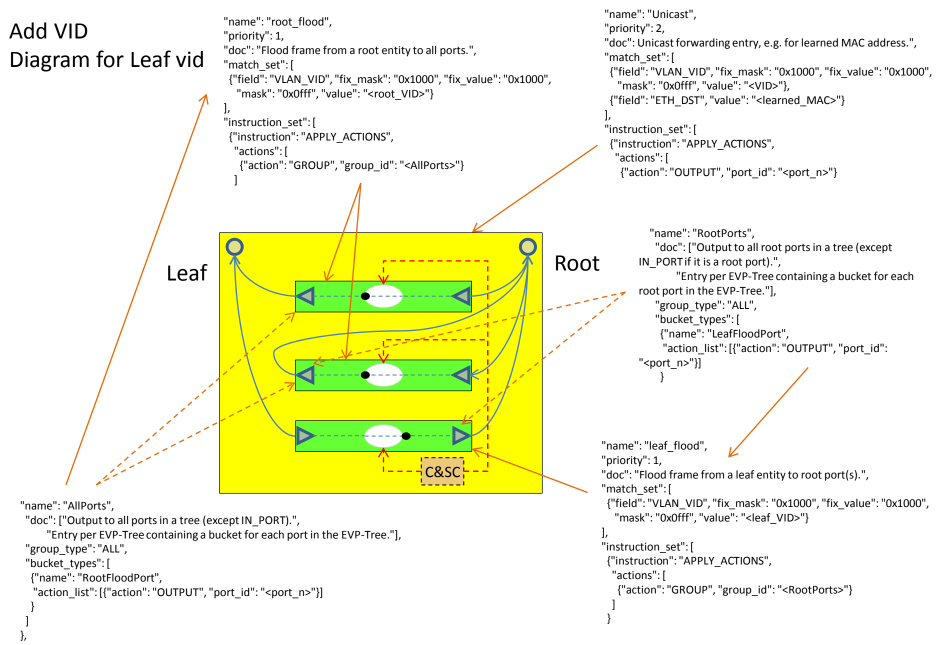


Figure 4-15 Sketch of mapping to OpenFlow using ForwardingSpec constructs

4.2 LogicalTerminationPoint and LayerProtocol specification

4.2.1 Rationale and requirements

The LTP/LP spec structure was developed to primarily support from a management-control perspective:

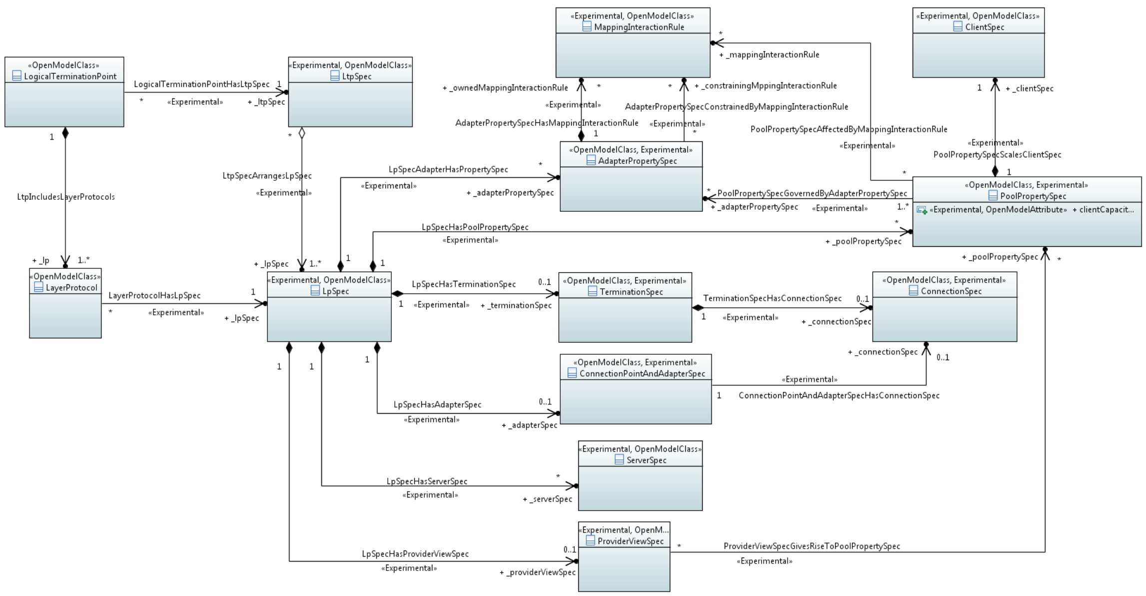
- The controlled introduction of layer protocol (network technology) specific attributes from sources external to ONF with minimum burden to ONF
  - ONF do not have a mandate for network technology (layer protocol) definition
    - Lesson: Work in other standardization activities has suffered from the burden of maintaining alignment between their redefined technology properties and the properties defined by the technology specification body
  - A method that provides ready access to the appropriate work of the other bodies whilst decoupling that from the ONF modelling work was considered vital
    - This eliminates the possibility of use of simple conditional composition
- The reducing of the definition of a network technology to match a specific realization

  - Often a specific realization of a network technology will not support the full technology definition
    - In many cases this reduce support will manifest via attributes that have reduced ranges etc compared to the standard
  - The method must support the narrowing of the definition of attributes within their current semantics
    - This eliminates the possibility of use of simple inheritance
    - The Pruning and Refactoring approach developed to enable generation of an implementation specific view of the Core Model has been used
- Specific rules that define the layer protocol mapping opportunities and also set out the interactions between layer protocols
  - This will allow the definition of the port layer stacks and link capacity etc
    - Layering is assumed and coded but to best enable innovative deployments layering should be explicitly stated
  - The method must provide a rich enough rule structure to allow definition of all currently understood layer protocol interactions
- Proprietary extensions to a network technology where the vendor has introduced a capability within that network technology
  - This maybe prior to standardization or for a niche application etc
    - This may involve extending attribute definitions within their current semantics
  - The method must have no barriers to such innovation whilst preventing unnecessary variety and ensuring appropriate decoupling
    - Attribute extension must be done in the context of the network technology outside ONF such that the bringing into the ONF context is always carried out by pruning
- Definition of a completely new network technology (that maybe proprietary)
  - This maybe a new or emerging standard
  - The method must guide appropriately the positioning of information related to the new technology to maximize consistency with other network technologies
- Migration and upgrade of definition of the technologies
  - The method should allow for on-the-fly redefinition
- Definition of constrained usage of a specific type of port
  - Where a capability already express by a specification is to be reduced for a particular application so that that reduced capability can be expressed by another specification provided through the view exposed to the user of that particular application

4.2.2 Model skeleton

The figure below provides a view of the structure of the LTP/LP spec model.

Once again, by modelling the internals of a LP, we can represent a 'prototypical instance' that can be used by LP and LTP instances.



CoreModel diagram: Spec-LtpCapabilitySpec

Figure 4-16 Class Diagram of the Spec Model of LTP and LP

The elements of the spec model are related to the pictorial symbols and hence functional blocks in the essential layer protocol model. The areas of the specification that support each aspect of definition highlighted in section 4.2.1 Rationale and requirements on page 27 should be apparent.

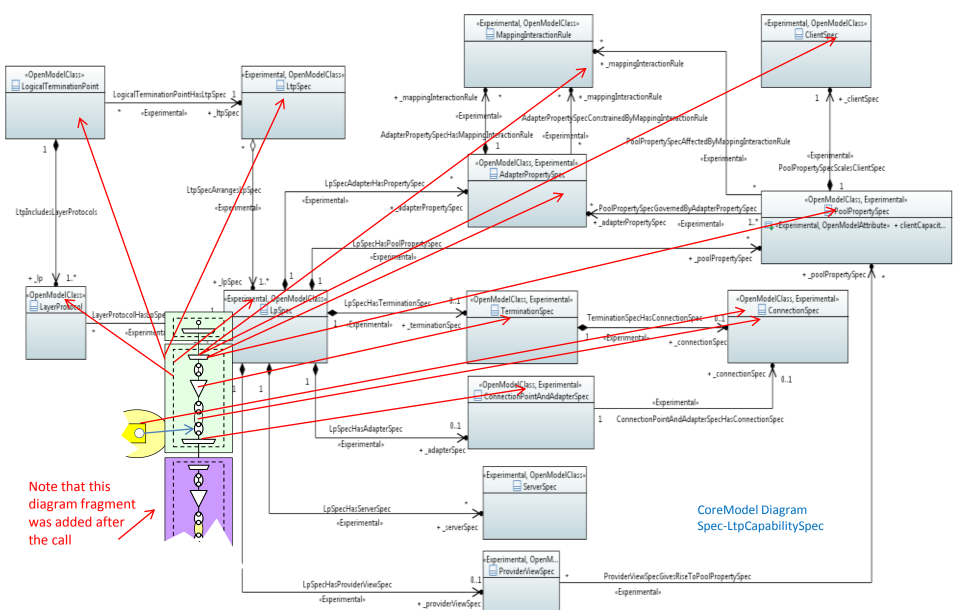


Figure 4-17 Relating LTP/LP spec elements to the pictorial symbols

The intention is that the LTP/LP spec model be capable of describing all cases of LTP/LP. Many cases are set out in [TR-512.2 ONF Core IM – Forwarding and Termination] (two figures, "LP Cases" and "LTP Cases", show the cases that the LP and LTP spec needs to support).

4.2.3 Class details

4.2.3.1 AdapterPropertySpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::AdapterPropertySpec

The specification of the properties of the client side adapter of an LP.

This class is Experimental.

4.2.3.2 ClientSpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::ClientSpec

The specification of a client layer protocol supported by the adapter of an LP.

This class is Experimental.

4.2.3.3 ConnectionPointAndAdapterSpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::ConnectionPointAndAdapterSpec

The specification of the server facing connection point and the adapter that deals with the transformation of a single signal of the layer protocol to/from the server. Equialent to an ITU-T CTP [ITU-T G.8052].

This class is Experimental.

4.2.3.4 ConnectionSpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::ConnectionSpec

The specification of the flexibility of the association between the ConnectionPoint and the Termination of the LP.

This class is Experimental.

4.2.3.5 LpSpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::LpSpec

The specification of the capabilities of a specific type of LP.

This class is Experimental.

4.2.3.6 LtpSpec

Qualified Name: CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::LtpSpec

The specification of a specific type of LTP.

This class is Experimental.

4.2.3.7 MappingInteractionRule

Qualified Name:
CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::MappingInteractionRule

The specification of the interaction between the support for different client layer protocols signals. For example an LP that support 20 layer protocol X signals and 5 layer protocol Y signals may be such that a particular layer protocol X instance being used eliminates the posibility of using a particular layer protocol Y instance being used.

This class is Experimental.

4.2.3.8 PoolPropertySpec

Qualified Name:
CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::PoolPropertySpec

The specification for the properties of the pool of available instances of a particular client layer protocol. This may cover numbering range, capacity, number of instances etc.

This class is Experimental.

4.2.3.9 ProviderViewSpec

Qualified Name:
CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::ProviderViewSpec

The specification of the properties of an LP at the base of an virtual/floating LTP that relate to the provider of capacity/capability for that floating LTP.

This class is Experimental.

4.2.3.10 ServerSpec

Qualified Name:
CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::ServerSpec

The specification of the server side of an LP at the base of an LTP that supports the creation of server LTPs for use in an inverse multiplexing scheme.

This class is Experimental.

4.2.3.11 TerminationSpec

Qualified Name:
CoreModel::CoreSpecificationModel::LtpCapability::ObjectClasses::TerminationSpec

The specification of the layer protocol termination (including framing, modulation etc). For example the specification of the function that takes a MAC frame and extracts the content (removing the MAC address in the process).

This class is Experimental.

4.2.4 Assumptions

The specification approach assumes that:

- For any particular layer-protocol there is a standard definition of a set of capabilities and that definition provides details of entities, attributes/properties and their values
    - The standard definition either is in Papyrus UML or can be transformed into Papyrus UML
    - The standard definition is properly layered and provides suitable guidance on whether a property relates to termination, adaptation, connectivity etc and whether it is a control property etc
- For any well-formed definition:
    - The classes and attributes/properties can be associated with the LP spec
    - The attributes of a layer-protocol each have unique names within the scope of that layer-protocol
    - Where there are multiple instances of a particular property then this property is contained in a class that has an association with an appropriate multiplicity to a class directly related to one or more parts of the specification

The above assumptions ease the work of ONF. Currently ITU-T provide definitions in Papyrus UML, other bodies do not at the time of publication of this document. Significant work to untangle some definitions will be required. It is highly likely that ONF will need to provide a service of migration of some layer protocol definitions into well-formed UML.

It is assumed that text documents that set out in loose form the functional dependency graph and related flow semantics will be required for a significant period of time to augment the specifications to enable development of code with significant behavior. The work here is for interface definition and driving of simple systematic behavior.

4.2.5 Broad usage

Just as the LTP/LP do, the LTP/LP spec model supports terminations that are:

- Ports bound to physical
- Floating in an NE
- Fully virtual in the network

The intention that any view of any termination with any degree of abstraction is supported (see TR-512.2 ONF Core IM - Forwarding and Termination and TR-512.4 ONF Core IM - Topology).

The figure below depicts the relationship between the various models and aspects.

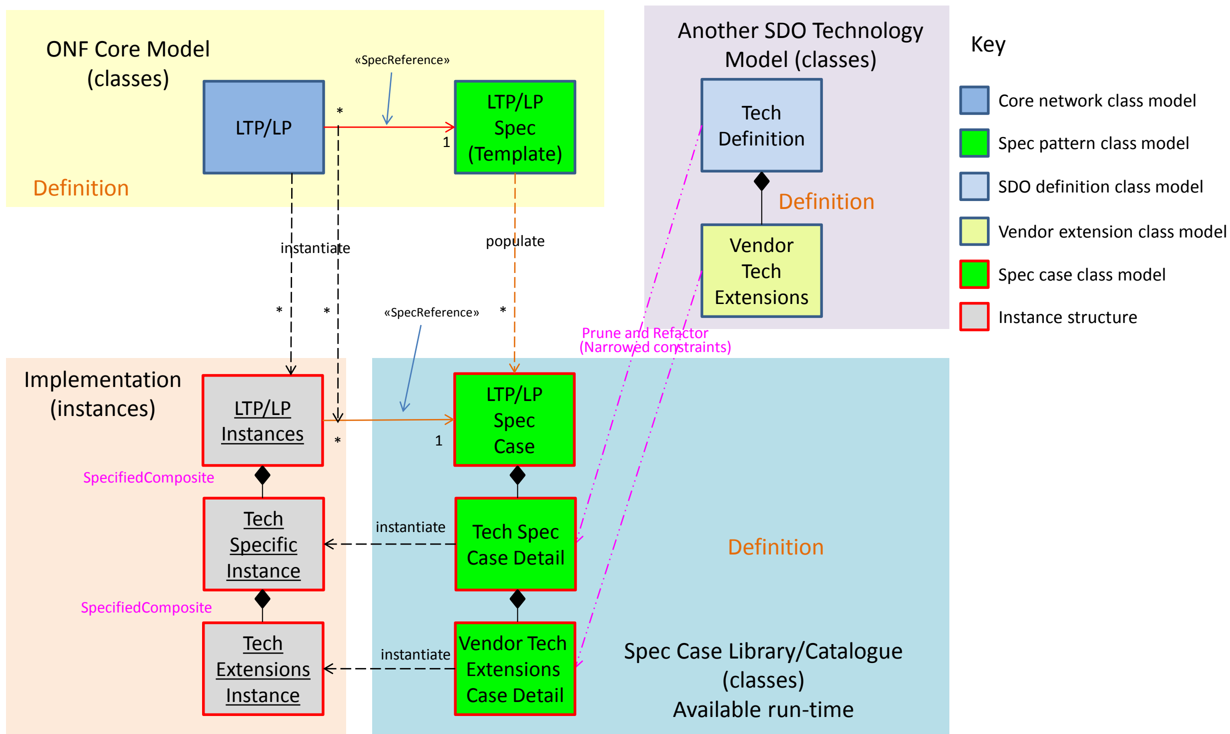


Figure 4-18 Relating LTP/LP spec with the class and instance models

In the figure above:

- The upper yellow area depicts the ONF core model.
  - The LTP and LP reference the spec model via an association with a stereotype to indicate that the association when instantiated will not point at instances but instead at a class model (schema)
- The upper purple area depicts a network technology model (e.g. as defined in [ITU-T G.8052]).
  - This model can be extended by vendors as necessary via a composition mechanism (perhaps via reverse navigation composition or technique favoured by the technology definition authority.
- The lower blue area depicts the integration of the specification model pattern and the appropriately pruned and refactored technology definition
  - The integration is achieved by using the spec model as a template
  - The spec model is cloned to provide a framework to fill in
    - The clone should be given a UUID to allow unique identification in the library
  - The appropriately extended technology model is duplicated (as a file copy to preserve the Papyrus IDs) and PruneAndRefactor associations are created between both the original model and the copy for all classes, attributes, associations etc in a mapping model[15]

[15] Ideally the Prune & Refactor tooling would be used for this. The tooling is in development.

- The attributes from the model copy collapsed into classes that reflect the structure of the spec
  - Highlight the essential association between elements of the spec and the elements of the model considering the potential challenges
    - Sub-layering approach
    - Models from bodies that do **not** split termination aspects (TTP/CTP for example)
  - Preferable that the model offers extension opportunities but this can be done by reverse association
  - Inheritance is expanded and containment collapsed based on the [1] and [0..1] rules such that a set of classes that reflect the basic pattern of the specification model leaving the [n..*] classes intact and any residual cross class associations intact
  - Attach the collapsed classes to the corresponding spec model classes via a [1] compositions
    - Note that the [1] compositions could be collapsed but the approach above maintains the mapping relationships in the most recoverable form
  - Attributes can be removed or pruned at this point
    - Each has ONLY aspects relevant for the case with ranges relevant to the case defined as modified data types and as necessary with OCL
      - Dependencies between items should be modelled explicitly including between attribute values etc – this should also be in the technology model and extensions
- The prune and refactor mapping associations should be updated at this point[16]
- The above process results in an LTP/LP spec case with appropriate UUIDs.
- For the runtime environment the spec case will be encoded in an appropriate run time schema language (e.g. Yang)
- The spec case is hence essentially a "class model" identifying all layering and attributes for a particular case of port etc

- The lower brown area depicts the instance model (obviously instantiation from some encoded schema form of the class model e.g. in Yang))
  - The structure and key attributes are derived from the class model in the upper yellow area (e.g. globalId for the LTP and localId for the LP etc)
  - Some LTP/LP instances reference the LTP/LP spec case constructed above using the UUID
    - Assuming that the LTP/LP corresponds to the specific case discussed above
  - The LTP/LP have attributes present that defined in the LTP/LP spec case (both pruned tech spec attributes and tech extension attributes) in addition to those attributes defined in the class
  - The controller can get the necessary specs from the library

[16] The Prune & Refactor tooling will provide some updates but at the point of delivery of this document the tooling is relatively immature.

The description above provides a general walk through of the operation of the LTP/LP spec model. The description was purposely simplified in some areas.

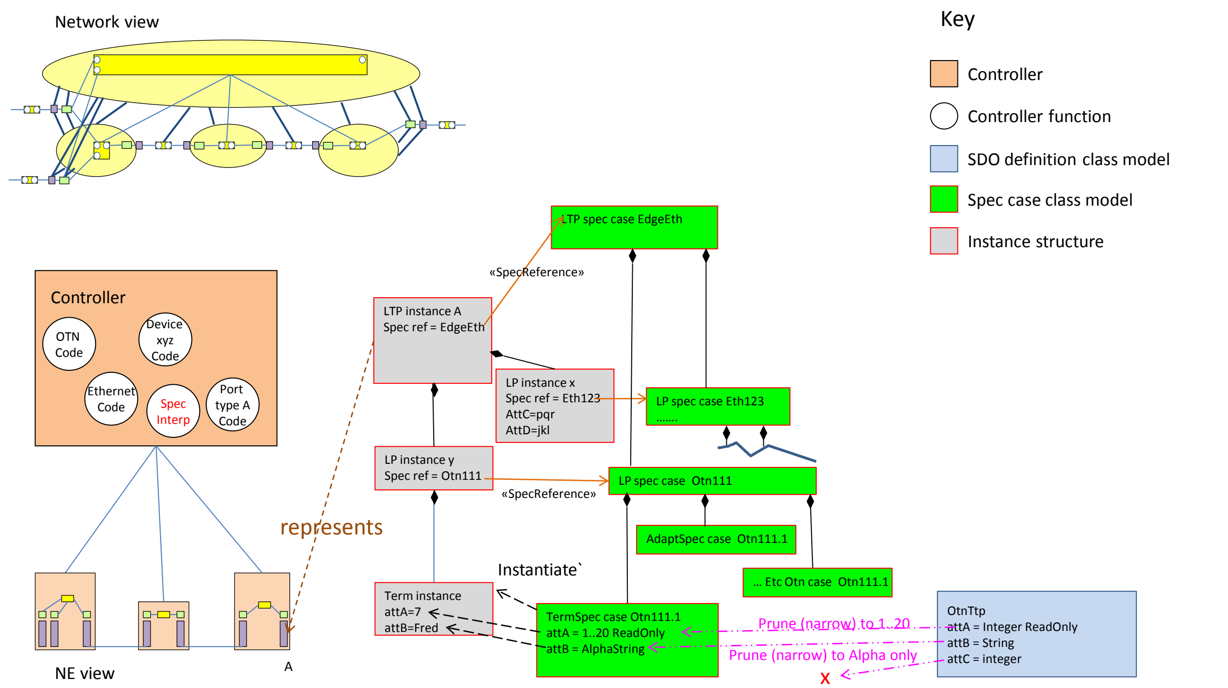


Figure 4-19 Sketch of use of LTP spec case

The figure above shows a sketch of the used of the LTP/LP spec (the color coding is as for the previous figure). To the left of the figure:

- At the bottom is a small network
- In the middle is a controller with various functions
- At the top is the controller view of the network

To the bottom right of the figure is a class of a technology spec with attributes attA, attB and attC. This technology class is pruned to form the TermSpec case Otn111.1 where attribute attA and attB are both narrowed in definition and attC is removed. The TermSPec case Otn111.1 is part of the LP spec case Otn111 which is itself part of the LTP spec case EdgeEth. The EdgeEth spec and its parts are available in the library.

Port A of the NE on the right is represented by the LTP instance A shown in grey and that has component parts defined by the LP spec Otn111. In the Term instance of the LP instance of LTP instance A there are attA and attB that are instantiations of the attributes in the TermSpec case Otn111.1 which hence abide by the definition of the attA and attB in the TermSpec.

4.2.6 Who constructs the spec?

The following roughly sets out responsibilities and ownerships.

- SDO X develops network technology model
  - Technology experts carry out the work

    - The model is structured as per the technology operation focussing on the specific layering of the technology
    - The work uses an approach:
        - That separates degrees of termination, adaptation and connectivity
    - The modeller identifies relevant optionality and what the rationale for support is
    - Note that any aspect of structure at this stage may be flattened in the implementation
- ONF provide examples of use of the spec model to describe the technology developed by SDO X
- Device vendors uses the process described in section 4.2.5 Broad usage on page 32 to construct necessary LTP and LP spec cases.
    - These cases are related appropriately per LTP/LP instance
- Service designer uses a process similar to that described in 4.2.5 to construct service endpoint definitions

4.2.7 Usage pattern

An LP composition per layer-protocol can be developed that has all multiplicity [0..1] and [1] held directly in the LP and that has all multiplicity [..*] attributes held in composed classes if the LP (essentially becoming a list form in a realization)

4.2.8 Spec example

The figure below provides a view of the detailed content of a partially formed spec case where the attributes shown were derived from [ITU-T G.8052]'

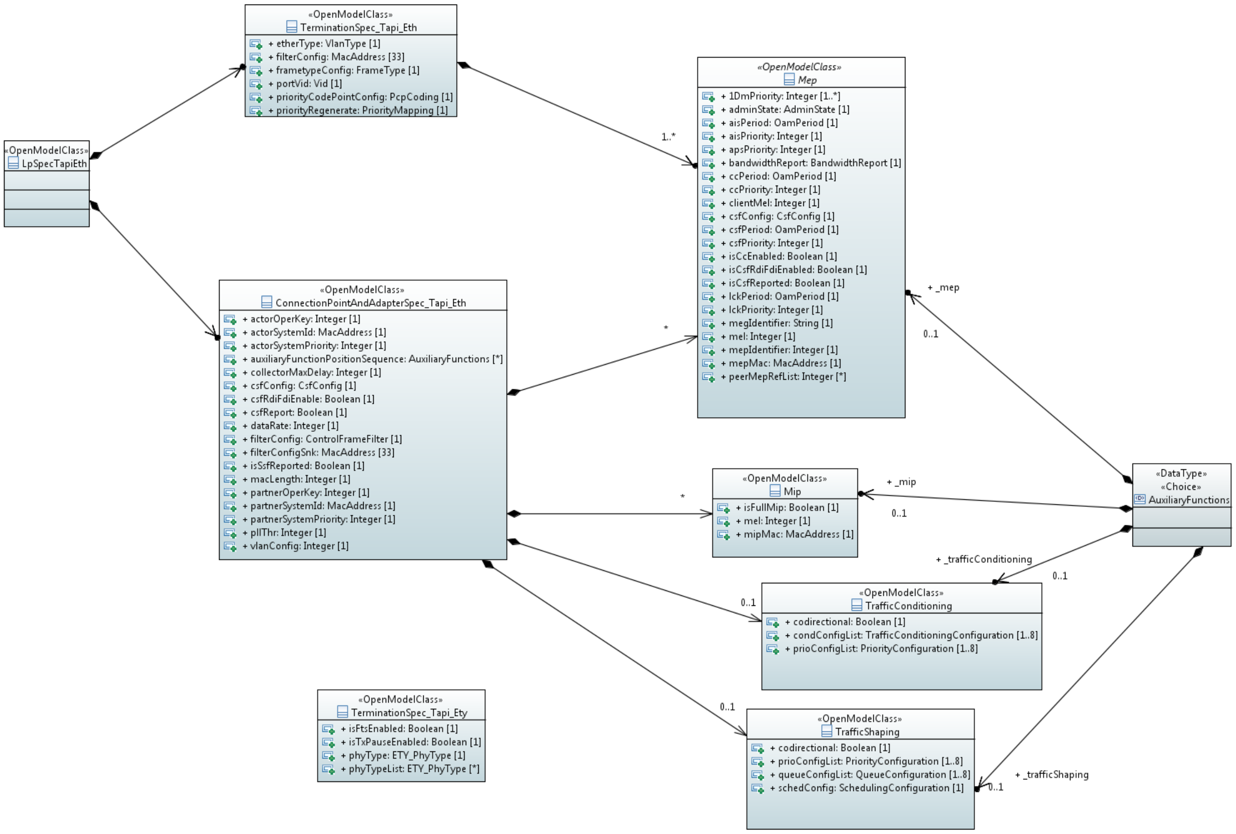


Figure 4-20 Sketch of Ethernet spec case

The following figure shows the Ethernet spec case in context of the "Full Layer, Fixed" LP Case (TR-512.2 ONF Core IM – Forwarding and Termination) and a fragment of the LP Spec model (see Figure 4-17 Relating LTP/LP spec elements to the pictorial symbols on page 29).

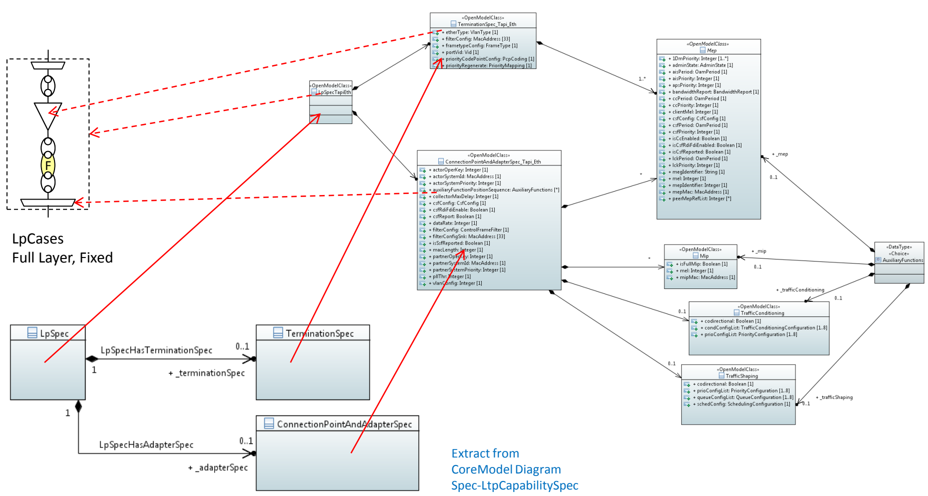


Figure 4-21 Sketch of Ethernet spec case in context of LP Case and LP Spec model

4.2.9 Running system

The following sketches the application of spec cases in a running system.

- Vendor populates spec cases (including equipment cases, LT cases etc)
- Vendor provides spec cases to the library
- Operator chooses to plan and prebuild intent for a particular type of "NE" (as part of a particular network fragment)
- "NE" spec identifies equipment spec which identify legal assemblies which identify instances of LTP cases supported
- Operator plans and prebuilds NE (expectation)
- NE is installed….
- Unexpected NE is discovered
- LTP specs acquired
- LTPs reported and validated against specs
- LTPs reported (not validated)
- LTPs to be used are validated for the relevant parts to be used

4.2.10 Adoption migration

The following sequence explores adoption and migration for the LTP/LP spec. This section develops from section 3.5 Adoption and migration considerations on page 12.

- Step 1: (a) Null spec (traditional approach) and (b) empty spec (first step to spec model)
    - Spec reference attribute is supported in LTP and LP but is either set to empty string or provides the name of a spec that when opened is empty (this is a legal spec and could be taken ongoing as an indication that a coded solution or a basic discovery solution is required)
    - Assumes traditional coded solution
        - Capabilities are identified during enrolment of devices derived from current live equipment
        - Capability knowledge may be pre-coded from paper specs for the devices
        - Any planning will necessarily be done using coded solutions or handcrafted equivalents to the spec model (proto-spec)
    - Solution form is similar to that for an [TMF 612] realization
        - Attribute list is flat per layer-protocol (similar to layeredTransmissionParameters of [TMF 612])
    - Sophistication level: Very Low
- Step 2-n: Rudimentary spec through to full spec
    - The spec is referenced but some parts are intentionally empty
        - It is assumed that at this level of sophistication all parts of the spec will be included but where the spec aspect is not supported the part will be empty (rather than absent)
        - For the parts not present traditional coding will be required
    - Expected growth path

        - Termination attributes included in the spec (connectivity and adaptation spec elements present but empty)
        - Connectivity attributes added (adaptation spec elements present but empty)
        - Adaptation attributes included (all properties are assumed to be independent)[17]
        - Inter-attribute and inter-adaptation rules included
    - Sophistication level: Low to Medium to high
- Indicating the degree of completeness/correctness
    - Use of lifecycle stereotypes in each spec case
        - Experimental/preliminary on an aspect of the spec means that that aspect in total will need code to support it and things will happen that are unexpected. Experimental means that definitions may be wrong as well as missing, preliminary means that what is in place should be correct (other than where specifically stated as experimental at the next level down) but there may be stuff missing
        - Example could be used in a general spec that provides suitable guidance but may have additional as well as missing attributes etc
        - Faulty could be used when an error is found in a spec (the spec is republished with the faulty item marked)
- Naming a spec
    - The class name is the spec reference
    - The subordinate classes can be named systematically as extensions to the spec name where the conditional pac in the entity would be expected to simply have the extension name (and be a structural item in the encoded form)

4.2.11 Various applications of LTP spec

The following figure provides a view of how various interrelated bodies would use the spec approach. Note that the primary ONF responsibility (not shown in the figure) is the formulation of the spec model.

[17] May want to find a way to indicate that some trial and error expected

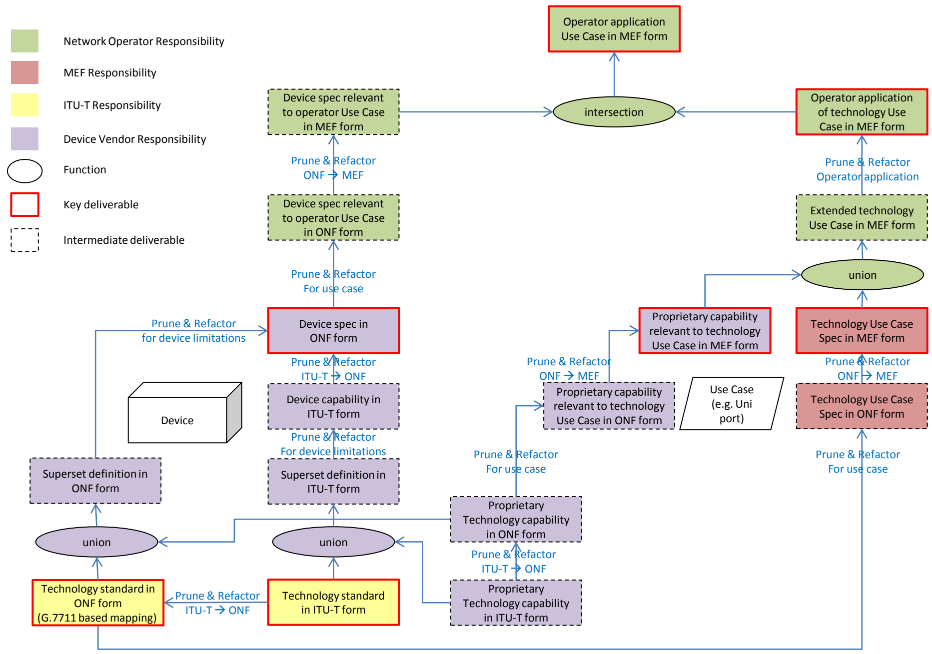


Figure 4-22 Sketch of spec usage

The flow starts with the Technology standard in ITU-T form at the bottom of the figure and via various transformations shows the production of:

- ONF forms of
  - Technology standard
  - Device spec
- MEF forms of
  - Technology Use Case spec
  - Proprietary capability relevant to Use Case
  - Operator application technology Use Case
  - Specific Operator application Use Case

As the aim is convergence of the MEF and ONF forms simplifications can be made to the diagram as in the figure below (where O/M form is the converged ONF/MEF model form).

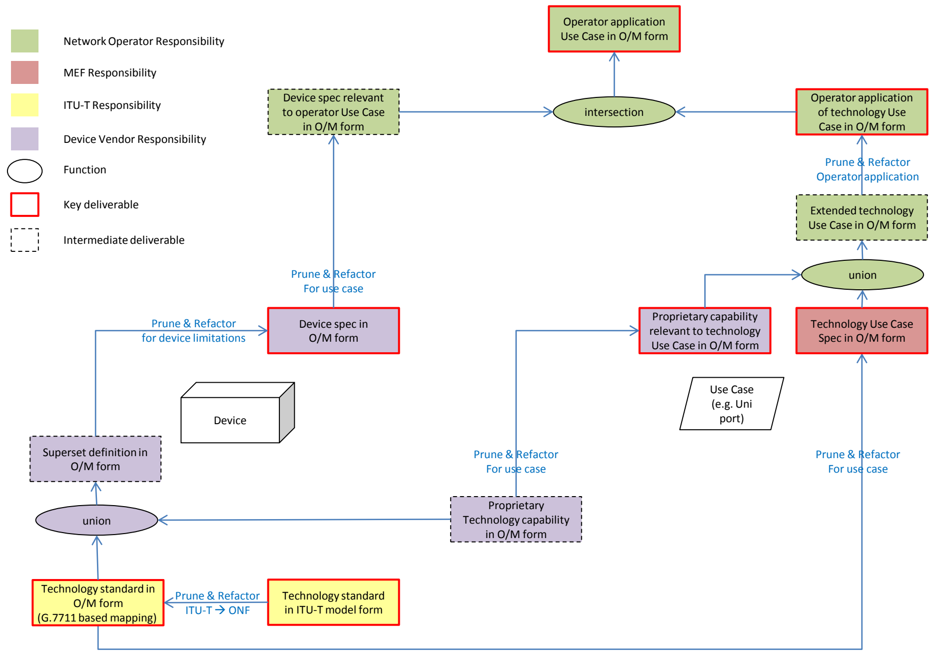


Figure 4-23 Simplified sketch of spec usage

4.3 ForwardingDomain (FD) and Link specification

The basic FD model assumes full flexibility such that there are no constraints on creation of FowardingConstructs) and such that the forwarding properties (such as cost, delay etc) are equal for all FCs created in the FD. The common case of Link is two pointed. Any multi-pointed Links are assumed to be fully flexible like the FD. In these simple uniform case (as discussed in TR-512.4 ONF Core IM - Topology) the FD/Link may have directly applied cost properties etc. However for an asymmetric multi-pointed case the properties will not be the same for all possible transits.

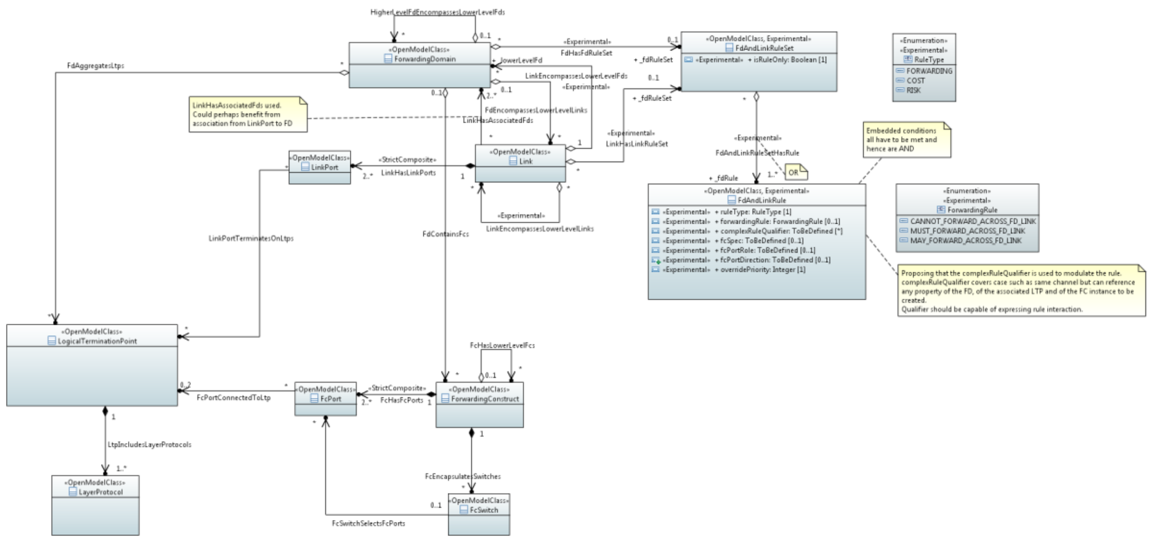
Where the properties differ for different transits and assuming that the FD/Link is "FC-opaque"[18] two options are available:

- Query per case: The client of the controller asks what specific properties would apply to a transit across the link/FD between specific bounding LTPs.
- Property model: The controller presents a property model for each FD/Link up front

This section discusses the "property model" approach. Presentation of a property model is assumed to be the most appropriate approach for general usage (including planning).

4.3.1 FD/Link rule/property model pattern

[18] The FD/Link does not decompose into subordinate parts for the purposes of creation of FCs.



CoreModel diagram: Spec-FdRulesInContext

Figure 4-24 Class Diagram of the Spec Model of FD and Link

The figure above shows FD and Link possessing rule properties. Some of the rule properties may be per case and others may be per instance; the same essential structure applies for both. For the symmetric case the properties apply to the FD/Link itself. Clearly the FD/Link can be broken down into subordinate parts (as described in TR-512.4 and depicted below in a figure for FDs from that document).

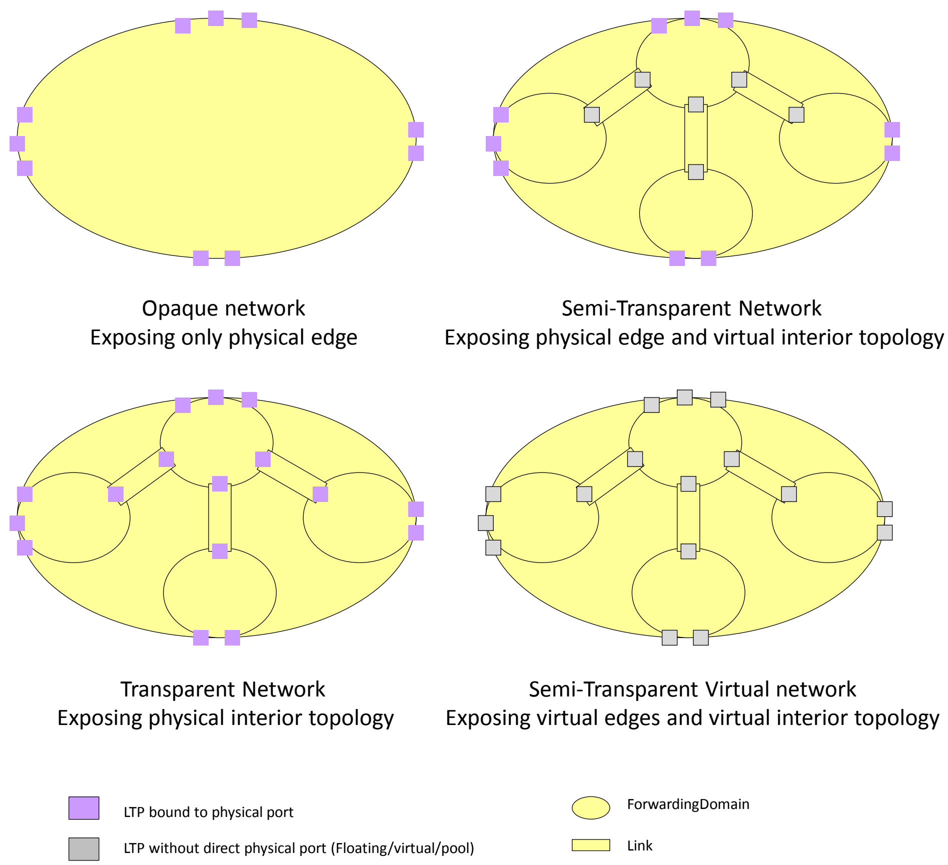


Figure 4-25 Various view boundaries

The subordinate FDs will have FCs present to represent the path of any top level FC across the subordinate FDs (as described in [TR-512.5 ONF Core IM - Resilience] and depicted below in a figure from that document).

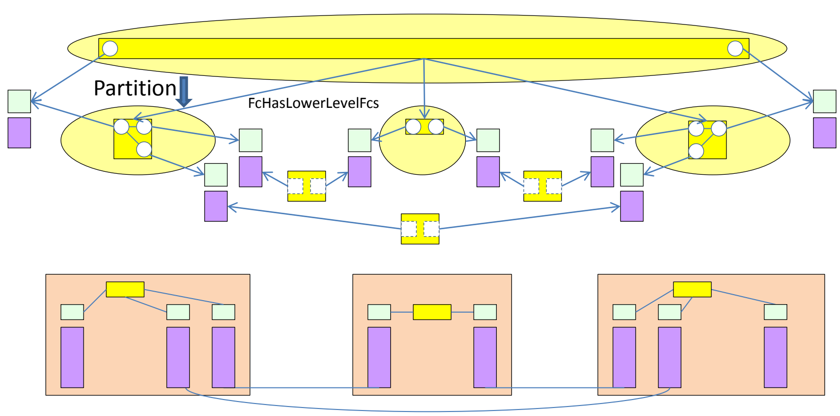


Figure 4-26 Simple summary example of 1?1 cases (represented via partition)

Clearly the rules related to building the top level FC in the figure above are provided by the partition FDs (where protection is allowed to be supported etc). For an FC-opaque FD there is no visibility of FCs in subordinate FDs but the FD could be partitioned to solely express constraints.

The constraint partitions need not be the same as the underlying FC partitions (in exactly the way that the apparent underlying FC partition depicted above need not reflect the real network as it may be an abstraction of the network – there is a brief discussion on views in TR-512.4).

4.3.2 Class details

4.3.2.1 FdAndLinkRule

Qualified Name: CoreModel::CoreSpecificationModel::ForwardingDomainAndLinkCapability::ObjectClasses::FdAndLinkRule

Set of "AND" rules related to creation of FCs across the FD/Link (i.e all rules have to be met for the FC creation to be allowed).

This class is Experimental.

4.3.2.2 FdAndLinkRuleSet

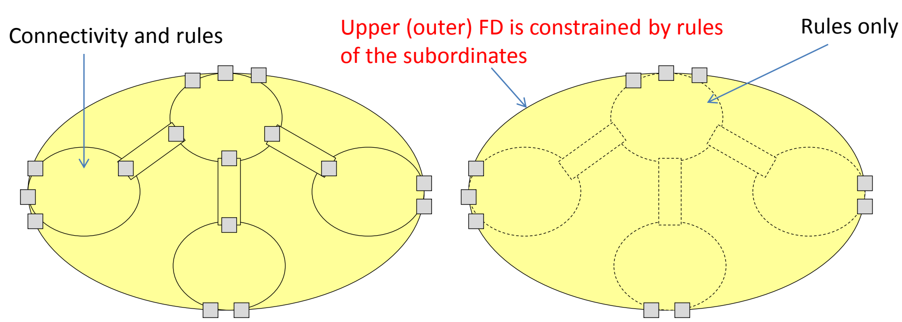
Qualified Name: CoreModel::CoreSpecificationModel::ForwardingDomainAndLinkCapability::ObjectClasses::FdAndLinkRuleSet

Set of "OR" rules related to creation of FCs across the FD/Link (i.e only one of the rules have to be met for the FC creation to be allowed).

This class is Experimental.

4.3.3 FD/Link rule model detail

The figure below builds on the earlier figure showing normal partition and rule only partition (the virtual edge form is used but this would apply to any of the depicted cases). The FDs that provide only rules are considered as LtpGroups (not explicitly modeled – the FD by its very nature is an LtpGroup[19]) and the Links that provide only rules are considered InterGroupRuleLinks (again not explicitly modeled).



[19] An explicit class may be developed for LtpGroup rules although other insights suggest that this would not be the right direction.

Figure 4-27 Normal FD/Link and rule-only FD/Link views

The method is based on the recognition that:

- At the lowest level of the real network there are deterministic elements the assembly of which provide the abstract view.
- At some level of decomposition of the FD/Link structure there is a simple description that can be applied.

The rule system operates as follows

- No rule statement at a higher level means refer to the next level down where rules at all levels are accumulated.
- It is assumed that a default of fully flexible would apply such that no rule statement at the lowest level means fully flexible, within the scope of that lowest level FD/Link.
- Rules within an FD correspond to LTPs bounding that FD
- Rules in Links correspond to LTPs in the bounding FDs only (i.e. FD-Link rules do not chain)
- Rules may be capability based or restriction based where a restriction overrides a capability statement
- Multiple different types of rules may be presented and should be analysed in combination where each type of rule should be in a single FD/Link partition (i.e. a type of rule cannot spread across several partition views) and multiple types can be in one partition

The figure below sets out a rough example of rules in a single partition (that was depicted in the earlier figure).

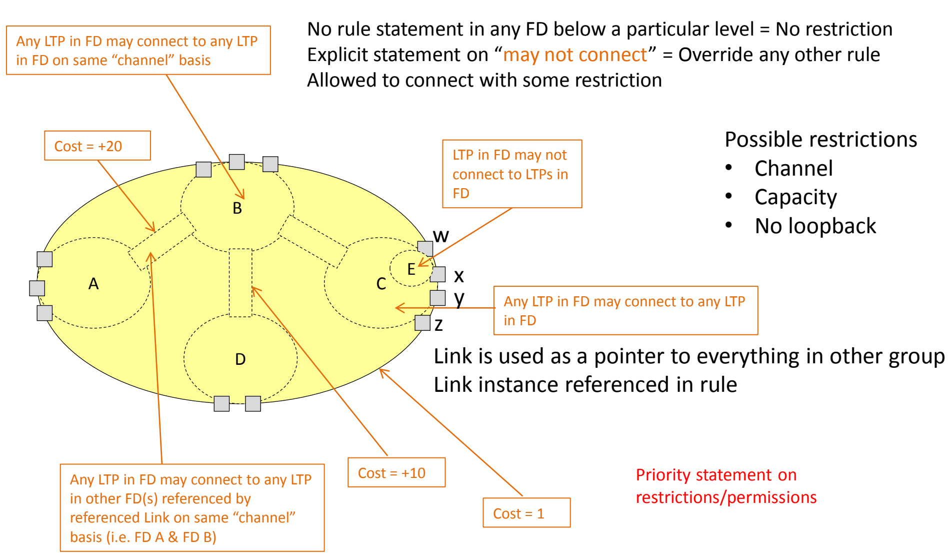


Figure 4-28 Rule example

Any FC created between exposed LTPs[20] must abide by the rules stated. The following assumes that bidirectional two pointed FCs are to be created (there are rules related to orientation of more complex FCs that are not discussed here). So:

- An FC can be created between any LTPs in A with no restriction for a cost of 1
- An FC can be created between LTPs in B so long as the FC uses the same "channel"[21] on the LTPs again for cost of 1
- An FC can be created between an LTP in A and an LTP in B so long as the FC uses the same channel on the LTP in A and the LTP in B and in this case the cost is 21
- It is not possible to create an FC from LTPs in A and LTPs in D
- In C, an FC can be created between LTP w and LTP y but not LTP w and LTP x (because E overrides other statements)
- An FC can be created between LTP x in C and any LTP in B with no restrictions for cost of 1
- An FC can be created between LTP in D and any LTP in B with no restrictions for cost of 11

There may be other constraints related to the network shown above. The figure below considers risk statements for shared risk assessment and relates them to LTP combinations using the mechanism described. The grouping for one rule set need not be related to the groupings for another rule set.

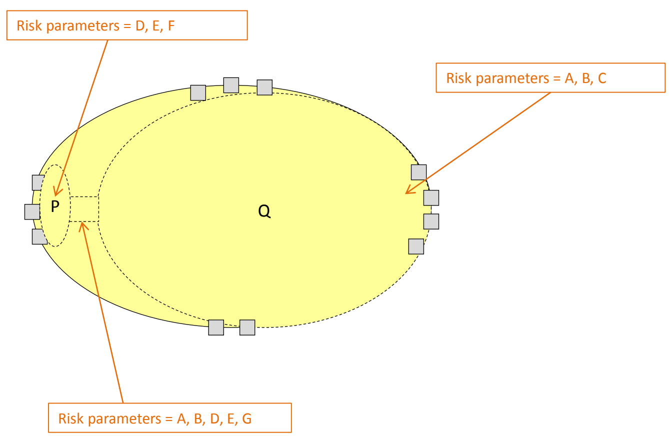


Figure 4-29 Rule example showing risk parameters

In the figure above an FC:

- Between LTPs in P incurs risk D, E, F
- Between points in P and points in Q incur risk A, B, D, E and G

4.3.4 Link asymmetries

The Link model follows the same pattern as the ForwardingConstruct and has the same need to deal with potential asymmetries, for example where a server layer FC offers dual homed

[20] Where the server LTP is represented at the boundary of the FD (see TR-512.4)
[21] Any property of the layer-protocol could be required to be preserved, channel is just an example.

protection to a client layer such that the server layer has four related ends. The figure below (adapted from a figure [TMF TR215] shows such a case where the link (highlighted with {DAD}) is multi-pointed asymmetric (and has exactly the same asymmetries etc as the supporting FC).

This complex case shows Interconnect ({IC}) protection with roles Resilient, Bridge and Protection and Double Add-Drop with roles (where the roles are in pairs (left M/S pair and right M/S pair)).

The lower diagram in the figure shows the HO path across the BLSR connecting the terminations in B, C, D and E that support the LO CTPs shown in the upper diagram in the figure below.

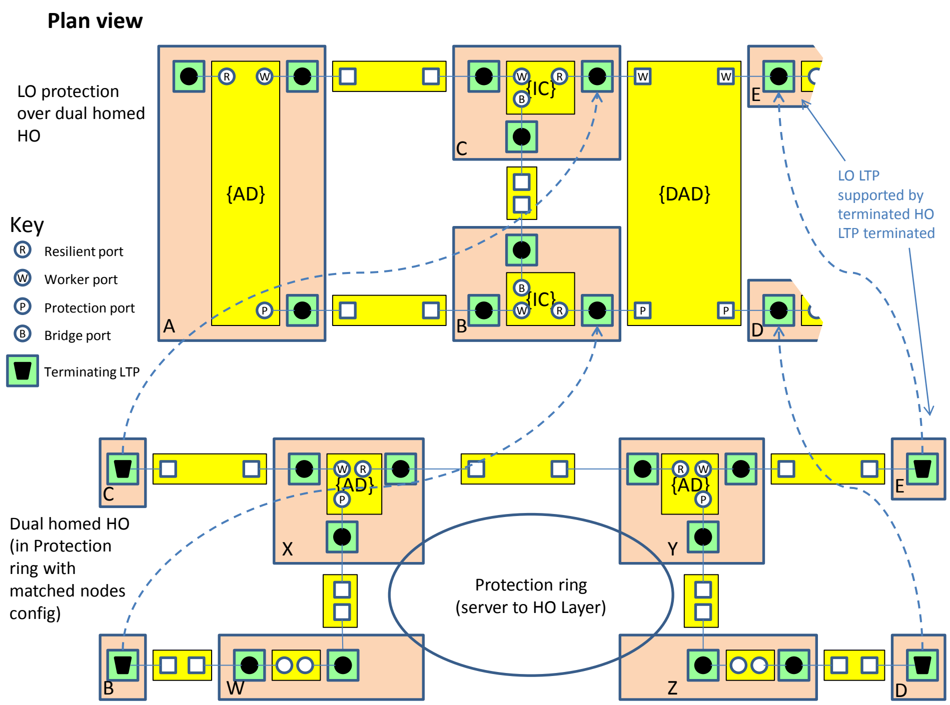


Figure 4-30 Multi-pointed flexible external FC scenario with Dual Homing

As shown in the earlier figure the Link has access to the ForwardingSpec.

4.3.5 LayerProtocol parameters

Also shown in the earlier figure the Link and ForwardingDomain have access to the LayerProtocolParameterSpec. This is a rudimentary model at this point. It essentially provides a vehicle to convey layer protocol specific parameters to the FC (and Link/FD).

There is a complexity here that has not been fully developed in that the parameters are invariable abstractions of properties of the LTP/LP. This will be developed in more detail in the next release.

4.4 Acquiring the specifications run-time

The following sketches show how a controller (or other system requiring details of the specifications) could acquire the specification on discovery of a previously unknown controllable thing.

The figures are intentionally generalized. The controllable thing could be a network element or another controller.

4.4.1 Initial system arrangement

The figure below shows a controller that has been designed to understand control of a layered network but that does not yet know any specific layer-protocols or parameters associated etc. The Controller is not yet aware of the Controllable thing, its instance repository is empty. There is a schema repository[22] known to the Controller.

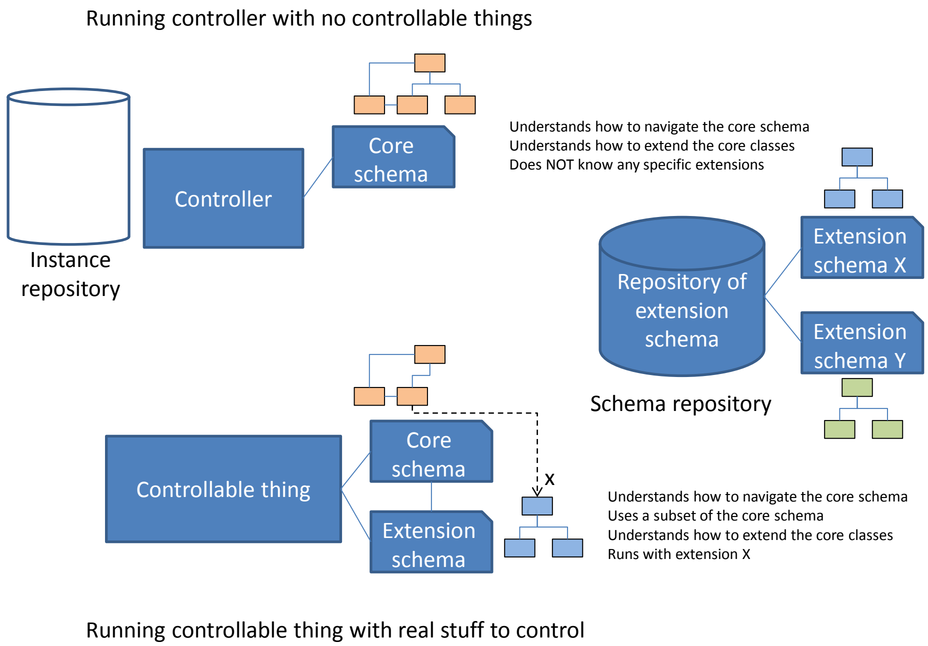


Figure 4-31 Controller and Controllable thing not yet connected

4.4.2 Learning to control the Controllable thing

In the following figure the Controller has been asked to control the Controllable thing. It has connected and is retrieving information.

[22] There may be many schema repositories (one per vendor) etc.

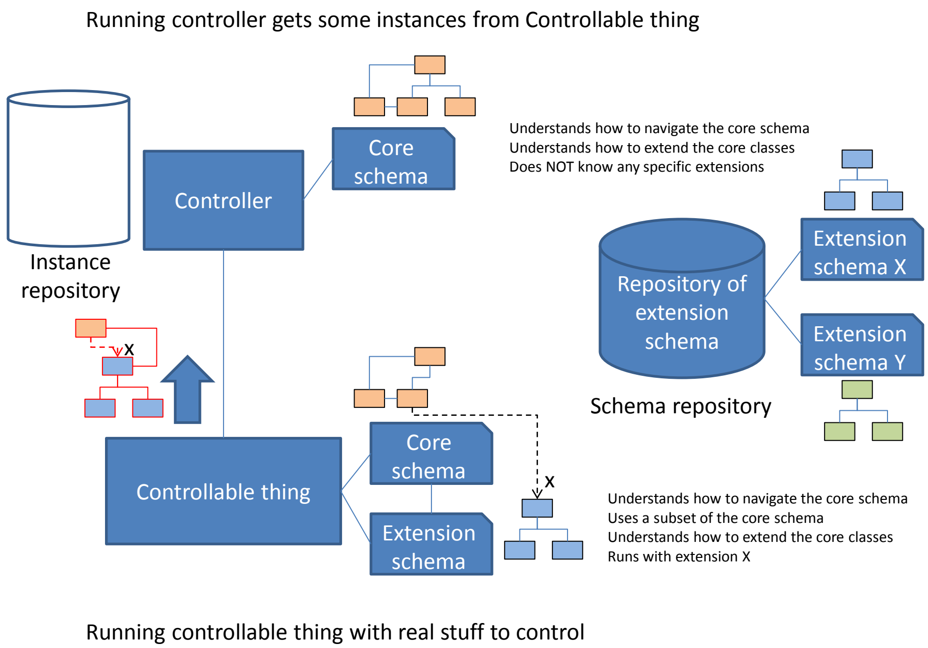


Figure 4-32 Controller connected to Controllable thing and retrieving information

The figure below shows that the Controller has stored the information it retrieved from the Controllable thing in its instance repository and has recognized that some information is not yet interpretable but that the information is related to a schema "X". The controller has asked the Schema repository if it has schema "X" and that schema is being sent to the controller. Schema "X" is a run-time version of a case of a specification (described earlier in this document).

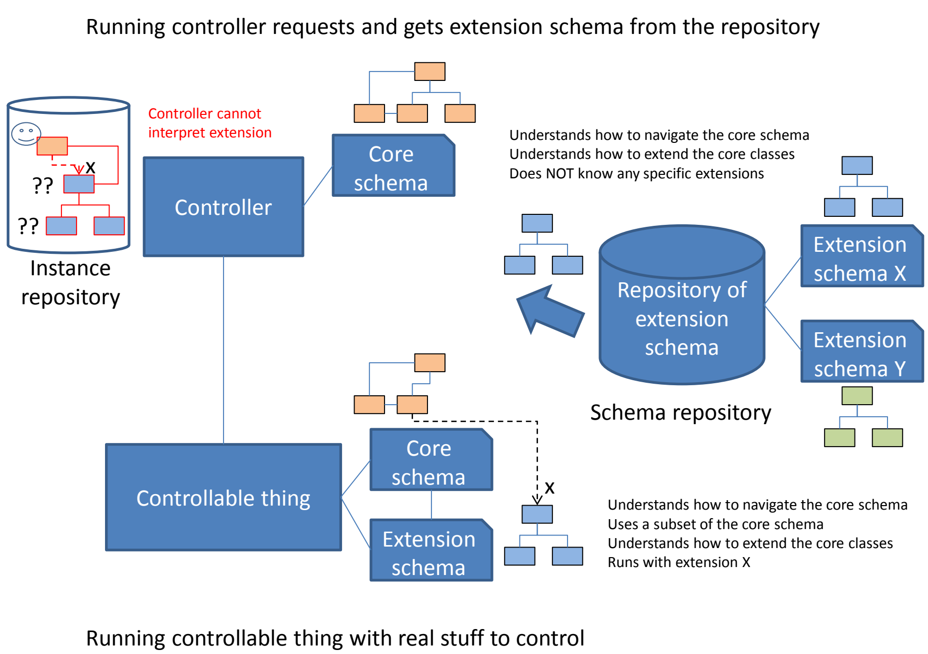


Figure 4-33 Controller acquires information on schema "X"

In the figure below the Controller now has schema "X" and hence:

- Has all invariant details for the discovered aspects of the Controllable thing (including information not known to the Controllable thing
    - These invariant details include simple information and rules
- Has the definitions of the dynamic properties
    - This includes read-only or read/write
    - Default values
    - Legal value ranges
    - Etc

Hence the Controller can interpret the attributes and in a basic solution provide the user with the opportunity to display and, as appropriate, set the attributes.

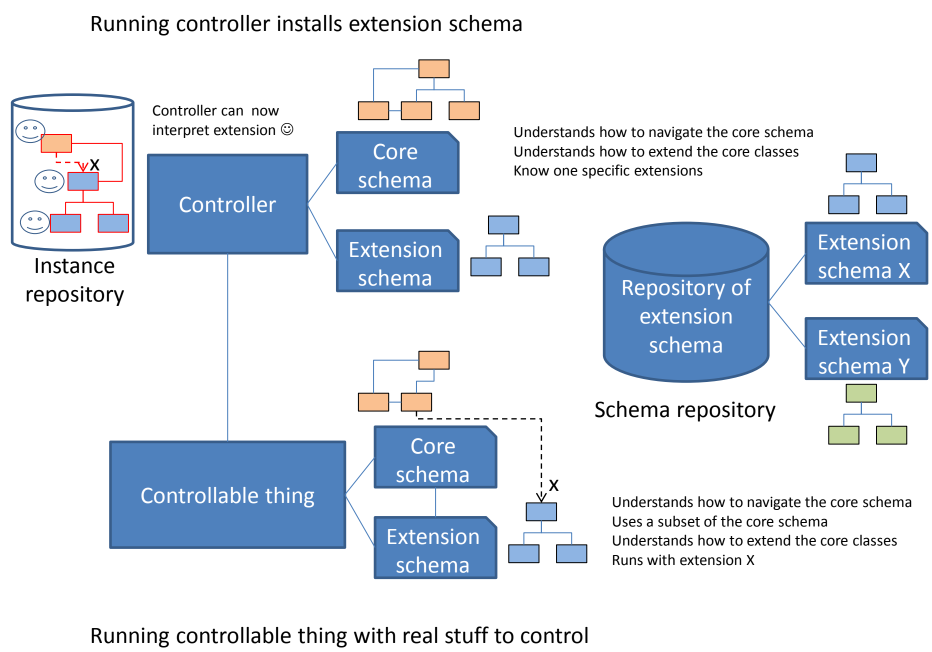


Figure 4-34 Controller can interpret and use attributes from schema "X"

4.4.3 Implications of the above

Clearly this is a very basic and simplified walkthrough and there are further system features required to fully integrate the Controllable thing.

- Controller API definition at start-up only includes the core schema
- Controller API definition is dynamically extended on arrival of a thing with an extension
- The controller may need to also acquire behaviour to deal with the new attributes etc of the extension BUT it may be able to provide significant capability without any additional behaviour

4.5 Work on the general pattern

This work is early draft experimental. This model fragment is not yet in the ONF Core Model and is provided here solely for information on direction.

4.5.1 Basic pattern

The figure below shows a sketch of the essential spec model pattern (Entity, Extension and Spec), adds a consideration of Profile and the relationship to the Spec and also provides a sketch of some potentially useful stereotypes, in the two extension specs to the top right of the figure, to deal with marking of properties that do not get instantiated in the entity instance.

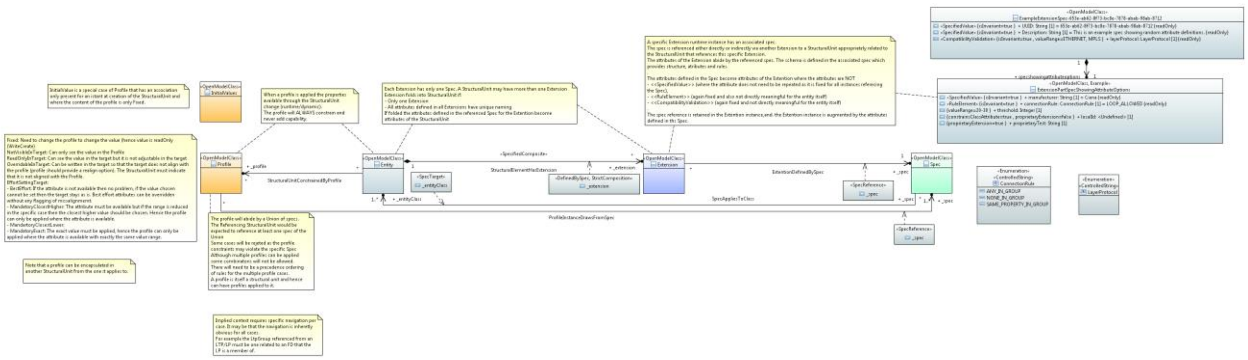


Figure 4-35 Basic spec pattern with example stereotypes

4.5.2 Rules related to the pattern

The following figure shows a sketch of the rules related to specification usage.

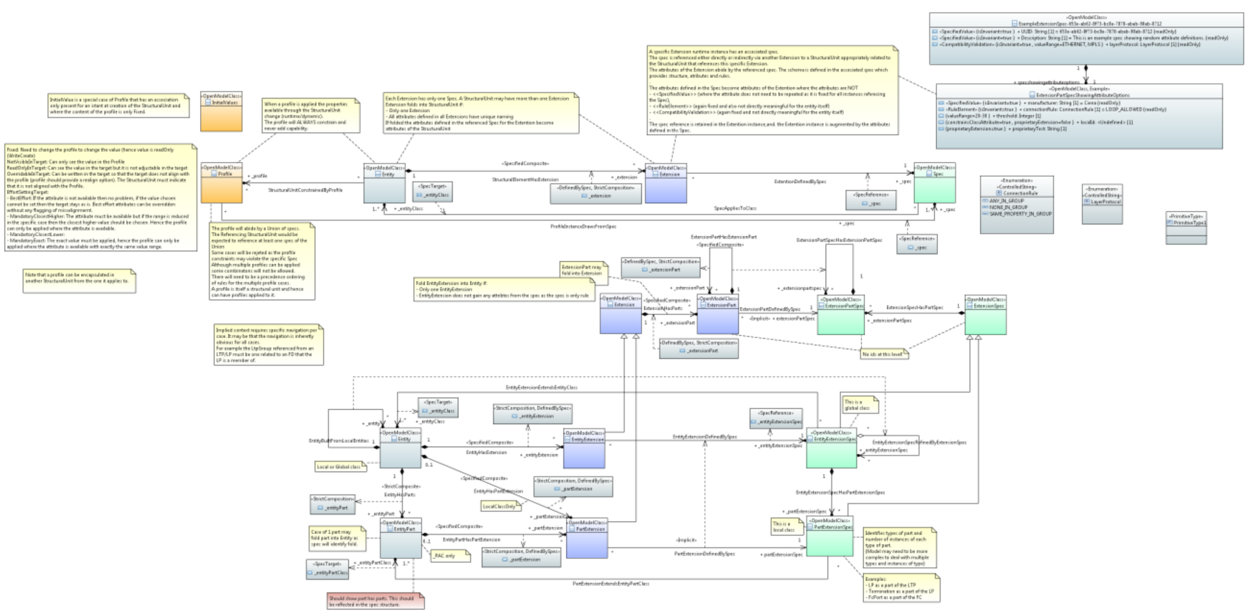


Figure 4-36 Basic spec pattern with rule sketch

4.5.3 Further vision considerations

Classification model becomes a rule pattern view of generalized (component-system) classes as per the sketch above

Any aspect of the rule pattern could be per instance or per case, invariant or dynamic.

End of document