Core Information Model (CoreModel)

TR-512.A.7 Appendix – Control and Interaction Examples

Version 1.4
November 2018

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

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
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
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

List of Figures

 © 2018 Open Networking Foundation

Document History

Version	Date	Description of Change
		Appendix material was not published prior to Version 1.3
1.4	November 2018	Initial version.

 © 2018 Open Networking Foundation

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.

© 2018 Open Networking Foundation

2 Introduction to this Appendix document

This document provides various examples of the use of the ONF CIM to model control and interaction between control systems.

The examples in this document are built from descriptions in other documents. The examples are supported by a combination of ControlConstruct, ControlPort (as described in TR-512.8) and ConstraintDomain (as described in TR-512.11) as well as ExposureContext (as described in TR-512.8) which is generally represented in the examples as a control domain.

In most examples, relationships between ControlPorts are shown as direct. In a majority of real cases, the actual relationship will be supported by normal networking which will be represented by FCs, LTPs etc. Some of example figures show an abstract FC, but in a majority of cases this has been omitted to reduce diagram clutter. Further information on the relationship between the Control Construct and the forwarding model can be found in TR-512.8.

2.1 Further context

The separation of concerns of Control from other aspects of a device is in line with the general trend to "disaggregate" functions. The ONF model provides a forward-looking representation of Control that can cover all device assemblies from the traditional simple "physical NE" through white box through virtualized to full cloud considerations.

2.2 Coverage in this document

Considering the wide spread of applicability, it is only possible to cover a limited number of cases in the examples at this early stage of evolution. These examples have been focused on current and near future solution needs. It is expected that further examples will be provided in later releases to better illustrate the breadth of applicability as the need arises.

The first few examples are for relatively basic "NEs" but later examples cover the generalized nature of control via more complex cases. The document works briefly through the ONF architecture team view of an SDN controller then embarks on a description of the use of the model to represent a more generalized control solution.

The current examples only explore the surface presentation of a control solution and do not dig into the essential control loop behavior behind the presentation. This deep view will be covered in later releases.

A majority of the the current examples in this document provide a static structural view. The Control model in TR-512.8 also begins to unpick interactions via the Operations model section which introduces the specific interfaces and the ControlTask and the via the Operations Pattern work in TR-512.10 which explores a generalize interaction pattern and messaging structure. Examples in section 3.3 Generalized Control Function on page 19 begin to tackle this area. More detailed consideration of messaging interaction will be covered in a later release.

© 2018 Open Networking Foundation

3 Control

This document uses a simple self-explanatory symbol set.

During the work to break apart the network element concept, the logical network functions were split off into ProcessingConstruct and ConstraintDomain. What was left was the network element control function.

The two things needed to represent the control function are:

- The (logical) location of control functions in the network and how they are related (control network)
- The scope of network functions[1] that each control function controls

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

It then became apparent that this general model could also be used to model other functional groupings e.g. an SDN controller, giving a consistent view of the different elements in the control network.

The text below will start with the device case and work up to the control network cases.

[1] i.e., the resources that the control function operates on. These could include any resources represented by FCs, LTPs, PCs etc.

© 2018 Open Networking Foundation

3.1 The Basic "Network Element"

On the left of the figure below is the representation of a simple 'device' as defined in the ProcessingConstruct document. Note, to keep the diagrams simple, ProcessingConstruct (PC) is used to represent all of the functions PC, LTP, FC, FD, SoftwareProcess etc.

ConstraintDomain (CD) is used to group the network functions and may also constrain them in various ways. For example, in the diagrams, a ConstraintDomain being used as a network element boundary is shown as CD=NE, a ConstraintDomain representing a physical chassis boundary is shown as CD=Physical(Chassis) and a ConstraintDomain representing a control domain scope is shown as CD=control domain.

On the right of the figure below, a ControlConstruct (CC)[2] has been added and another ConstraintDomain to represent the scope of control (control domain). The ControlConstruct and ConstraintDomain are related by an association in the model "CdConstrainsControlConstruct".

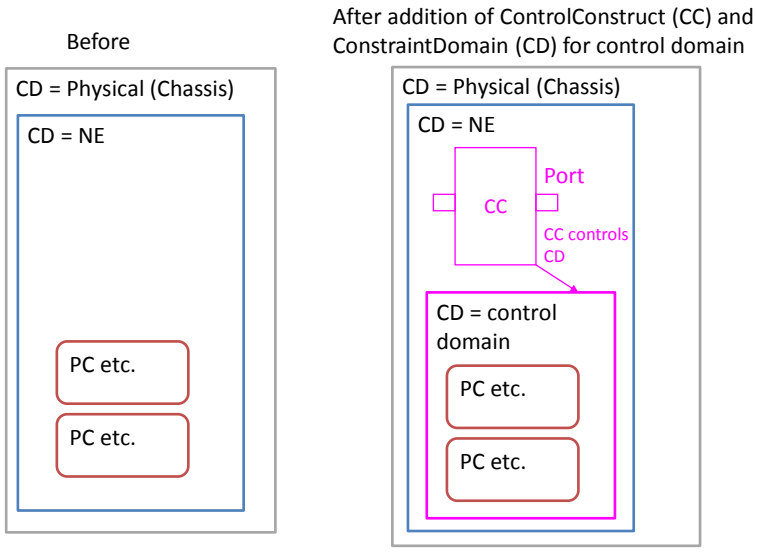


Figure 3-1 – Basic Network Element

[2] See TR-512.8.

© 2018 Open Networking Foundation

The next step is to be able to show a control network. ControlConstruct (CC) has ports (using the component-port pattern) and the ports can be bound together to show the logical binding. An attribute in ControlPort is used to show the type of relationship, client-server, master-slave or peer-peer. Also, because ControlPort is associated to LTP, it can be related to any transport functions of interest (see TR-512.8 for more details).

The control function layering within a constrain domain is represented by having a ControlConstruct inside of a ConstraintDomain (CD) that is controlled by another ControlConstruct as shown in Figure 3-2.

Note that we should not allow control loops or ControlConstructs to control themselves. This is not represented in the model and would be enforced by the constraints attached to the ConstraintDomains.

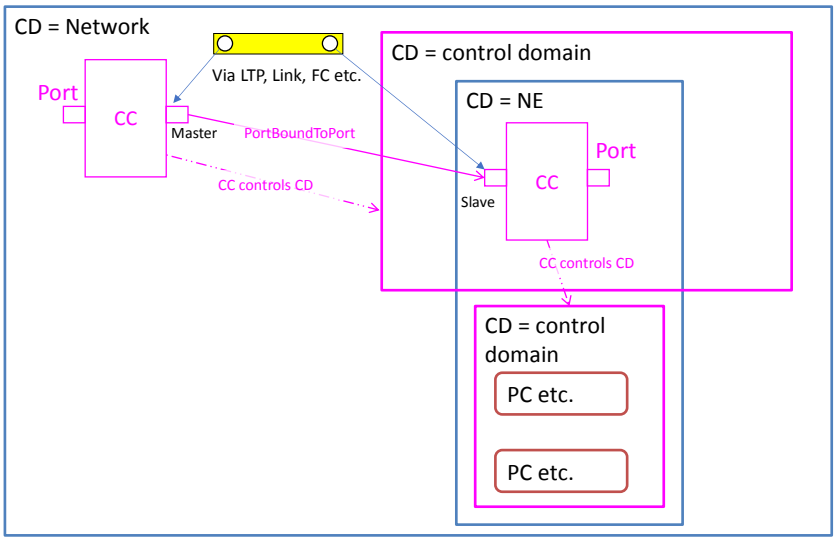


Figure 3-2 - Basic ControlConstruct layering Use Case

The instance diagram below shows how the example above can be represented in the model

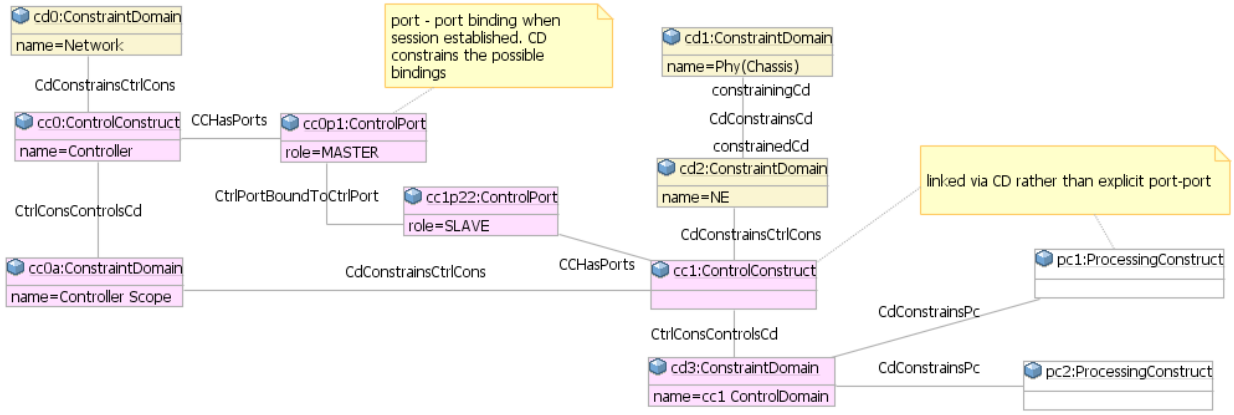


Figure 3-3 - Basic Use Case

© 2018 Open Networking Foundation

It is possible to add control ports[3] to every network function, as shown in the figure below, and then bind these to the control construct ports.

This makes sense architecturally and provides a nice consistency, but:

- Locally within a NE, the binding is usually implied rather than explicitly defined and managed (e.g. we define a BGP process **via** the ControlConstruct so its binding is implicit)
- It adds a lot of complexity to the instance graph, to create and manage all these ports and bindings
- Since we expect some sort of local management agent, the bindings are local, so the transport between the ports is not modelled i.e. FCs are not used

So it is recommended not to instantiate the network function control port bindings within a device, but to rely on the implied binding from the ControlConstruct (CC) to the ConstraintDomain (CD) that it is controlling.

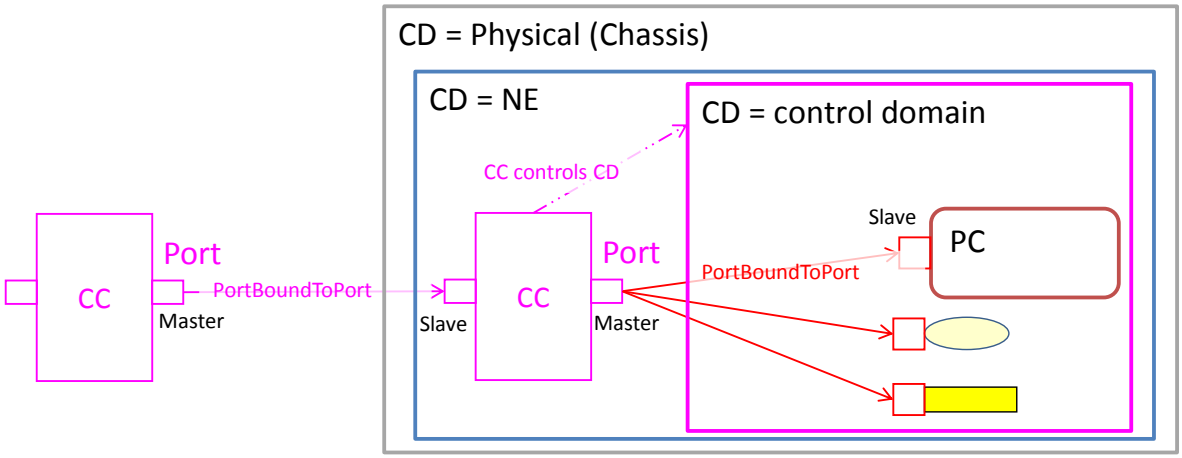


Figure 3-4 - Control port to PC port binding

The remainder of this section describes how other common, but more complex, cases can be modelled.

The model is not limited to supporting just these cases, but it is not practicable to try and cover every possible case.

By covering the general partitioning and aggregation cases, it should be easy to determine a suitable representation for other cases.

Note also that we haven't covered 'virtualization' here, but the same principles apply.

For an example of ControlConstruct controlling software, see TR-512.A.13.

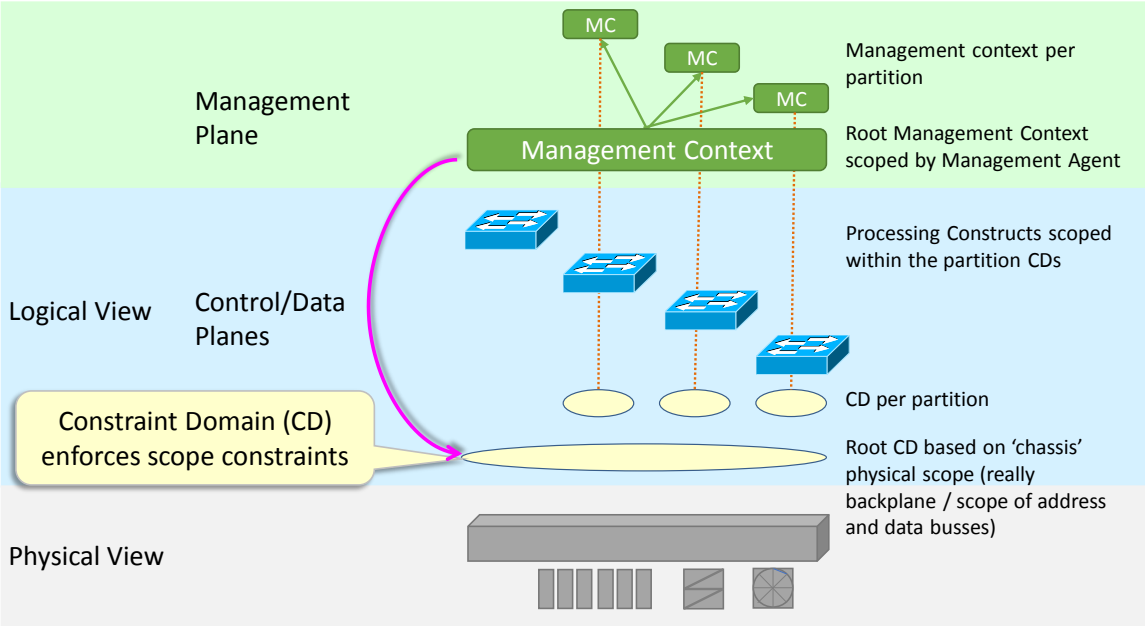
[3] See TR-512.A.2 for a discussion on the Component-System pattern and a view of the generalized Component showing Operations port for the control of the component.

© 2018 Open Networking Foundation

© 2018 Open Networking Foundation

3.1.1 Physical device partitioned into more than one logical device

A common case is where a physical device can be partitioned into more than one logical device. This may be done in a number of ways, with varying degrees of partition autonomy. Note that there may only be one physical management agent, but it is likely that each partition will appear to have its own logical management function.



The management plane may be global or partitioned, or both (as shown).
Root MC, Root CD and Physical Inventory have same scope.

Figure 3-5 - Device Partitions

There is no need to change the model for this case, all that is needed is to create the required ControlConstructs and ConstraintDomains and to relate them.

© 2018 Open Networking Foundation

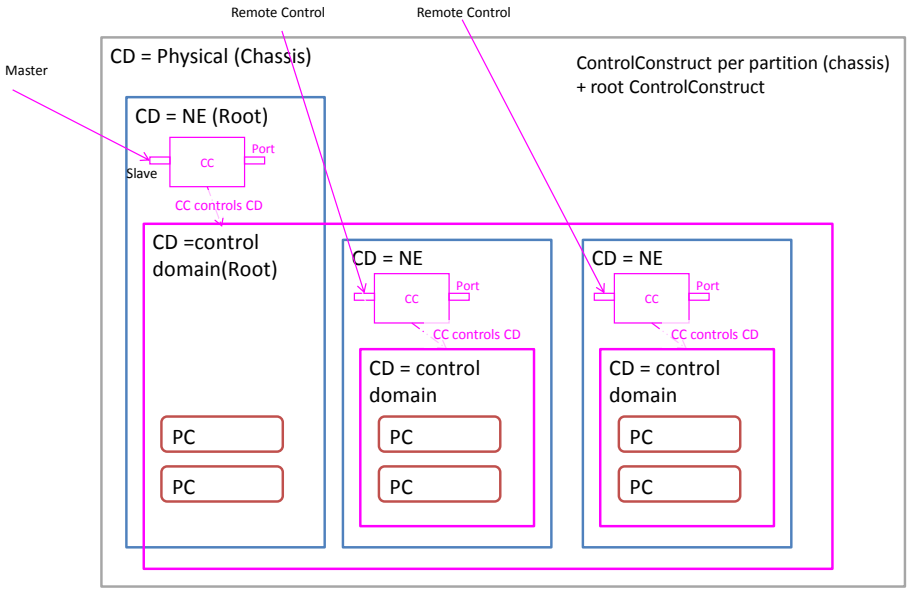
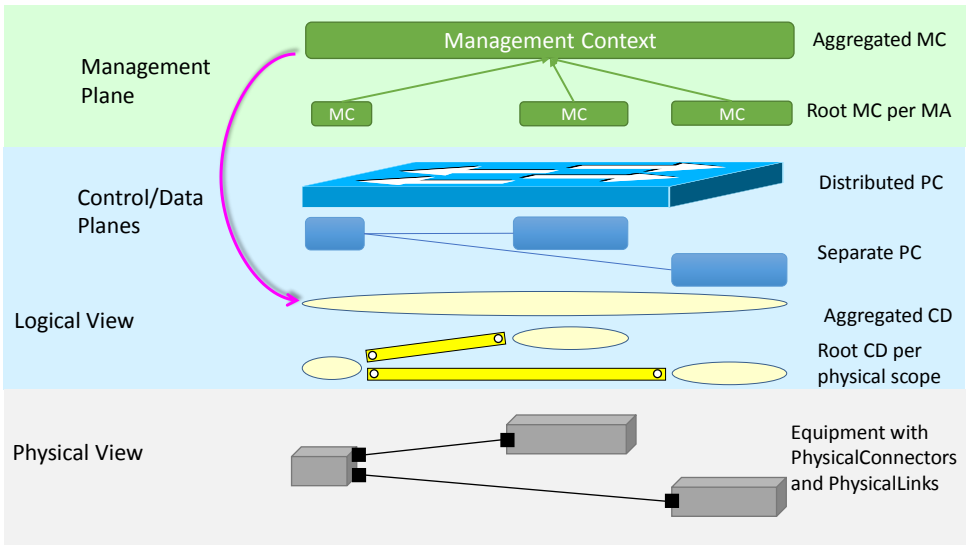


Figure 3-6 - Device Partitions Model

3.1.2 Multiple physical devices aggregated into a single logical device

Another common case is where many physical devices are aggregated to behave as a single logical device. There may be a number of variants on this and two of these are explored below.

In this example, we will consider a single logical device where each physical device is managed separately. Here MC = ManagementContext and MA = Management Agent.



The management plane may be global or partitioned, or both (as shown).
Root MC, Root CD and Physical Inventory have same scope.

Figure 3-7 - Distributed Device – Separate MA

© 2018 Open Networking Foundation

Again, the same model represents this case. Note that there may be distributed ProcessingConstructs that cross the physical device boundaries (such as PC-3 in the diagram below).

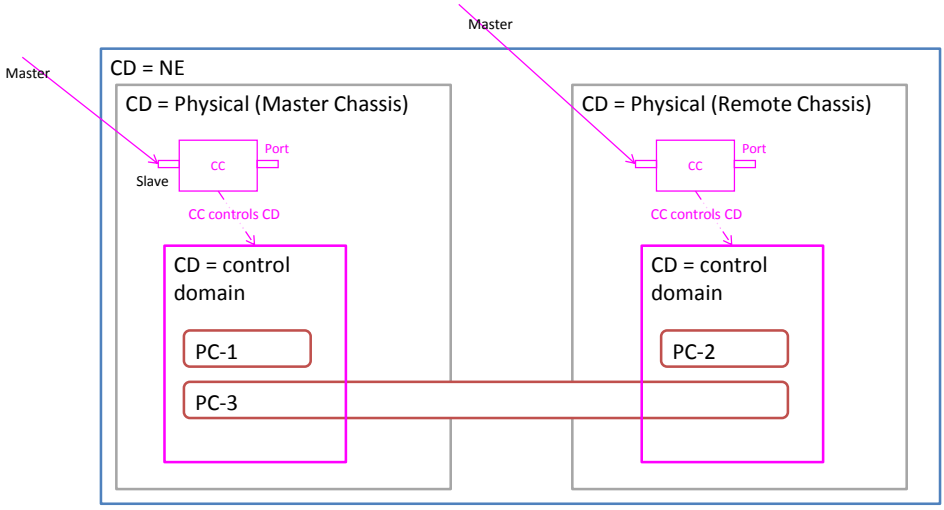


Figure 3-8 - Distributed Device – Separate MA Model

© 2018 Open Networking Foundation

In the single management agent option, the main difference is that there is only one management access point and the remote chassis ControlConstructs are slaved from the master chassis ControlConstruct.

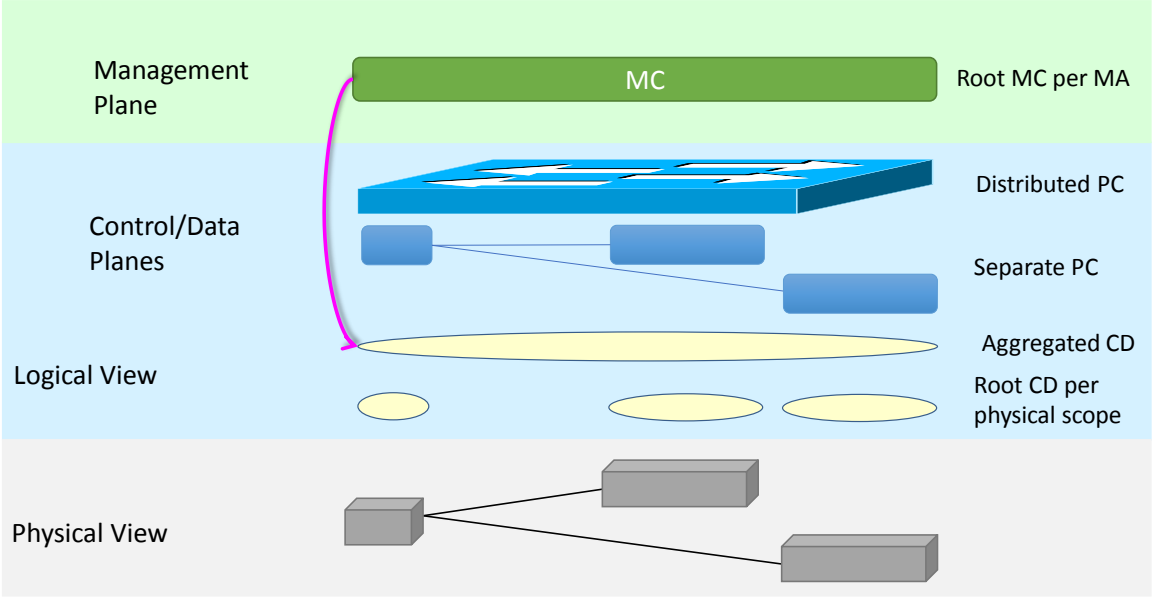


Figure 3-9 - Distributed Device – Single MA

Again, we see how the general model elements can be arranged to support this option too.

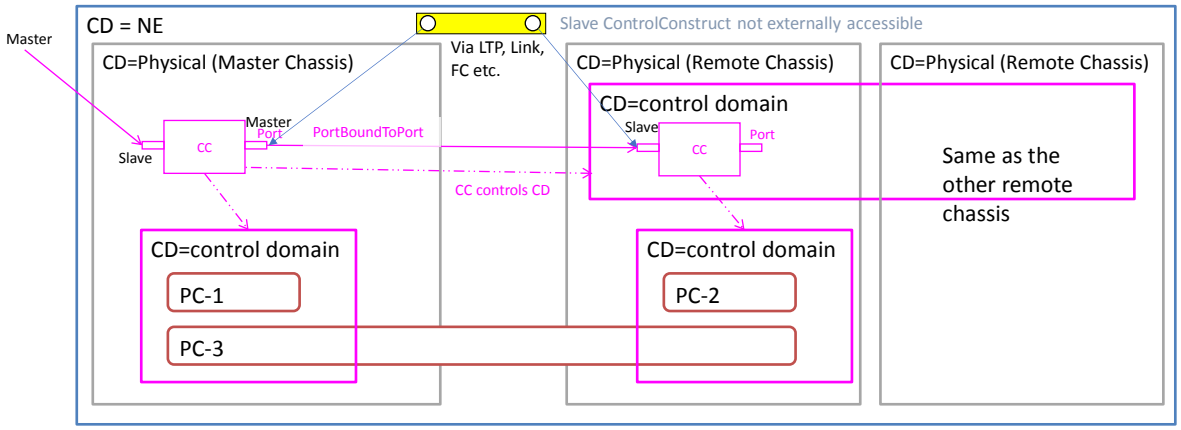


Figure 3-10 - Distributed Device – Single MA Model

© 2018 Open Networking Foundation

3.1.3 Hybrid partitioned and distributed

The last case, which does occur in real life, shows the versatility of this approach without having to resort to odd workarounds. In effect it is a hybrid of the partitioned chassis and the distributed device cases that were covered earlier.

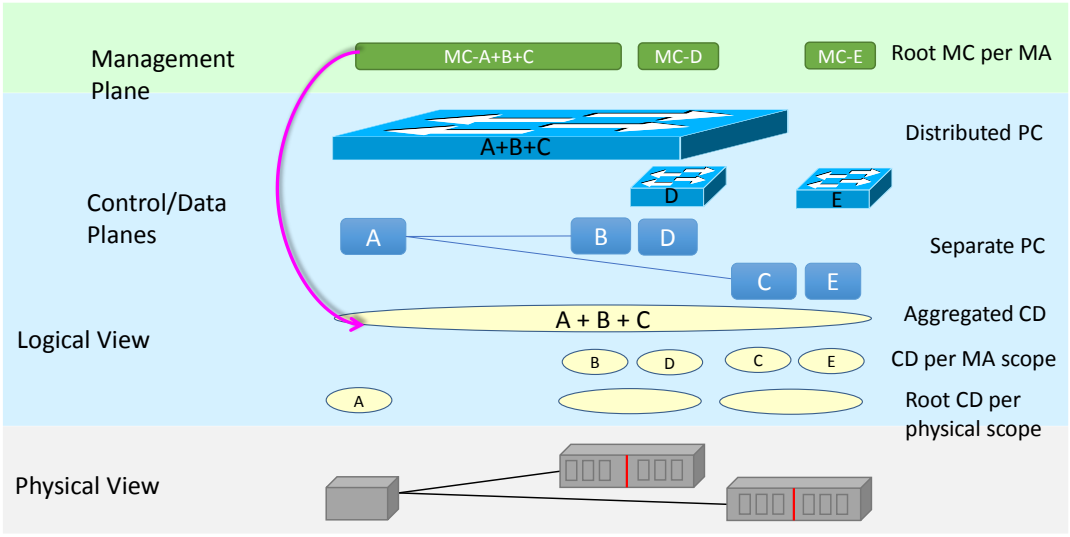


Figure 3-11 - Distributed Device – Split Chassis

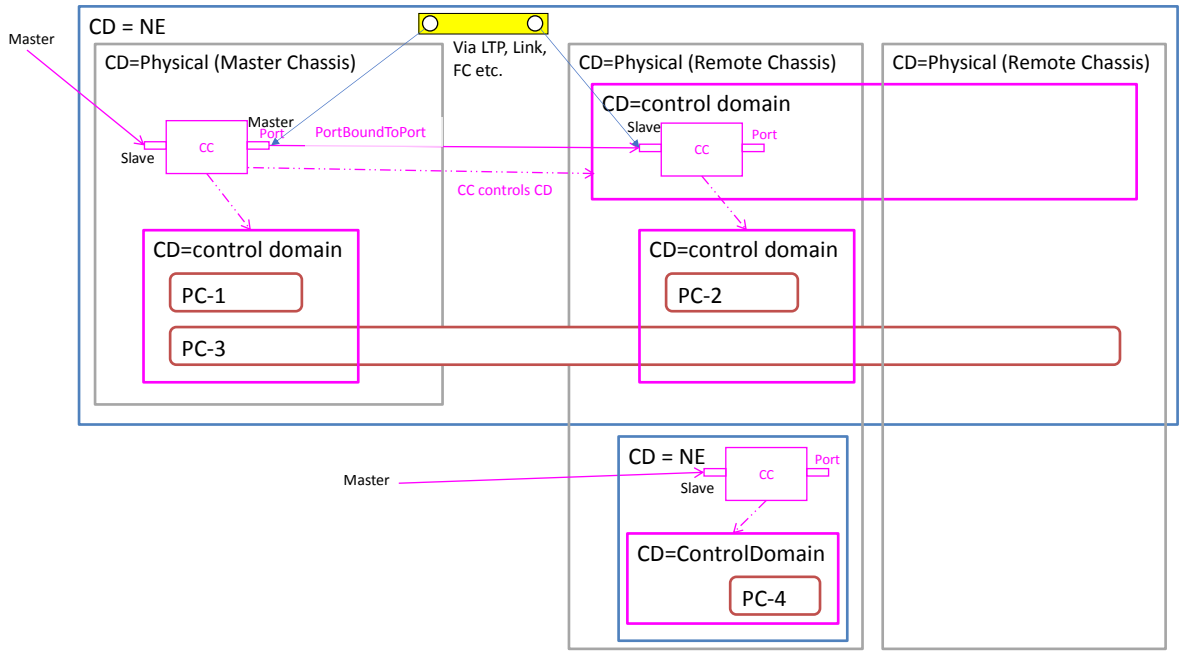


Figure 3-12 - Distributed Device – Split Chassis

© 2018 Open Networking Foundation

3.2 ONF SDN Controller

The ONF architecture document [ONF TR-521] defines an SDN controller structure as shown in the figure below.

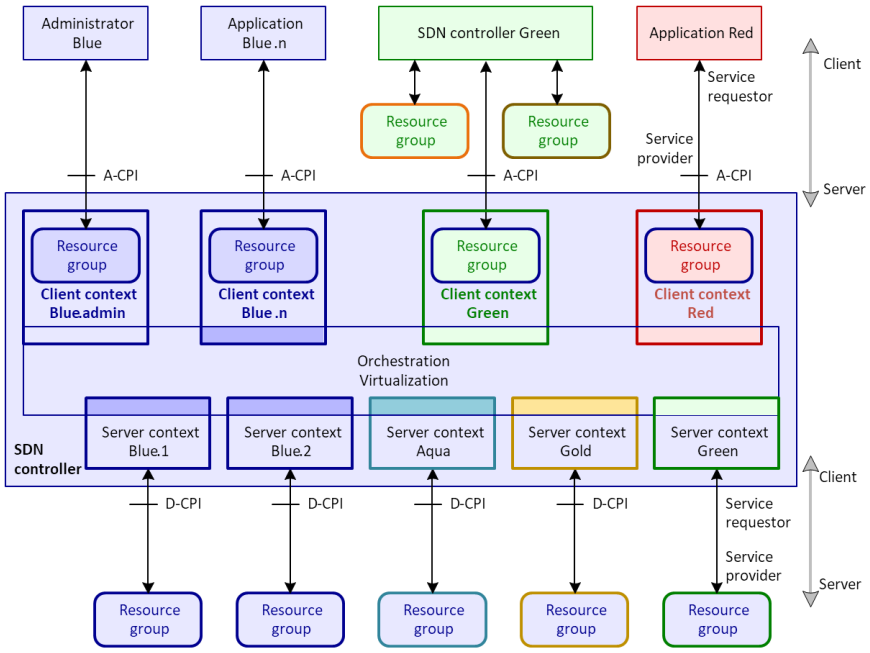


Figure 3-13 – ONF Controller Architecture

The rest of this section will show how the control model can represent this case.

ConstraintDomain is used to represent ResourceGroup so no additional class is needed for that concept.

ConstraintDomain is also used to represent ClientContext and ServerContext.

© 2018 Open Networking Foundation

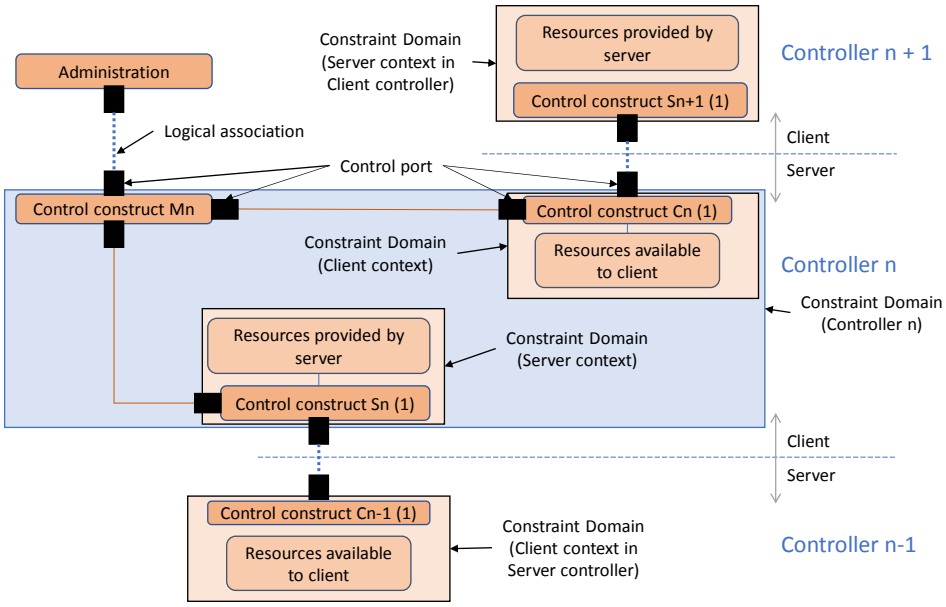


Figure 3-14 - ONF Controller Architecture Example

Note that there is a ControlConstruct *instance* per ServerContext (S) and per ClientContext (C), giving a total of 1 (for administration) + S + C ControlConstructs.

The ControlConstruct Mn in the diagram above, will need S + C ports to bind to the internal ControlConstructs, plus any ports required for external access. The model in Figure 3-14 models the controller architecture of Figure 3-13 and can form a controller hierarchy. A controller can support multiple server contexts and multiple client context.

 © 2018 Open Networking Foundation

3.3 Generalized Control Function

The model doesn't constrain how control functions can be assembled into a control network.

In this section a possible representation of a general control function will be explored.

We will assume that a ControlConstruct can *control* many other ControlConstructs, with a dedicated port at each end.

We will also assume that a ControlConstruct can *interact* with many other peer ControlConstructs[4], with a dedicated port at each end.

We will also assume that a ControlConstruct can only be *controlled* by a single other ControlConstruct (ignoring high availability for now)[5].

A general control function:

- May transport / route / switch other devices' control / management messages (packets)
- Is different from the router / switch / transport data plane because it produces and consumes (control / management) messages (making it a semantic content endpoint like a CPE / host)
- The control / management messages from network devices become the controllers data plane messages (a relative concept which can be represented using roles)[6]
- Control of the controller is done via its ControlConstruct and the controller would consider these to be control / management messages

First, we will define a ManagementContext (MC) as a type of ConstraintDomain that can enclose a ControlConstruct.

A ManagementContext scope could be:

- A physical device
- A Virtual Machine (VM)
- A software container
- A software process

Now we will define a ConstraintRequest (CR) as a request that is in terms of constraints and a ResourceResponse (RR) that is in terms of network functions (PC, FC, FD, LTP, Link …).

A ControlConstruct can send and receive both ConstraintRequests and ResourceResponses.

[4] In a peer relationship the role bias will be defined for each interaction, transaction, session etc.
[5] This does not preclude another controller controlling the underlying resources via some other access. This is simply a statement of access to control the things in the view presented via the ExposureContext. This does not preclude one controller controlling the things controlled by another ControlConstruct via one ExposureContext and another controller controlling the ControlConstruct itself via another ExposureContex. This does not preclude changes of role.
[6] Another way of looking at this is in terms of the Component-System pattern as set out in TR-512.A.2. A Component has ports related to its purposeful behaviour and ports related to control of its behaviour (Operations ports). A ControlConstruct has the purpose of controlling other components, hence its purposeful port have the purpose of controlling other things. These are connected to the Operations ports of other Components.

© 2018 Open Networking Foundation

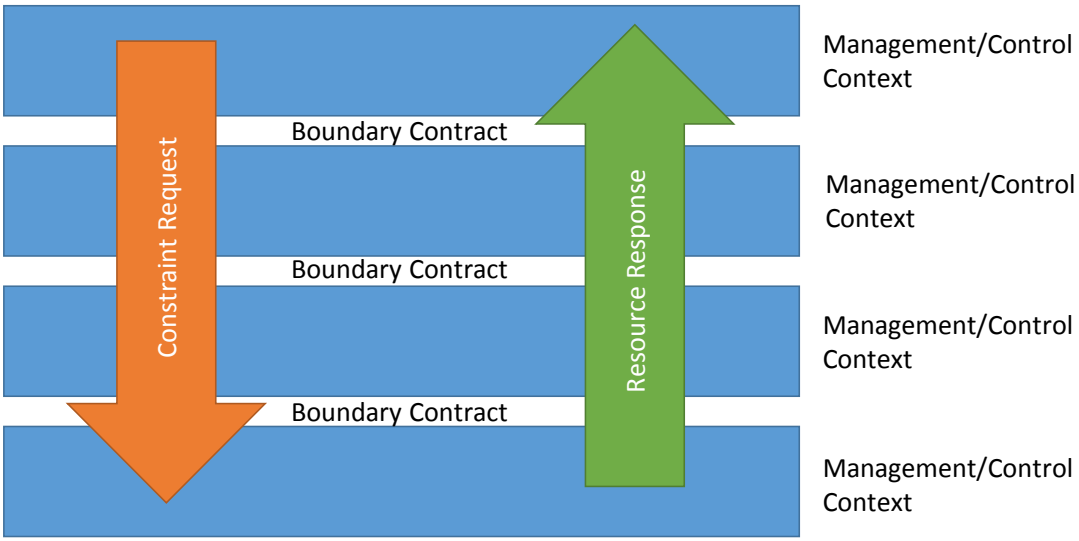


Figure 3-15 – ConstraintRequest and ResourceResponse

Constraint Requests originate from Needs and ResourceResponses are designed to fulfil the Need. The request does not necessarily have to propagate to the bottom management context as it may be fulfilled before then.

The control relationship doesn't need to be hierarchical, for instance it could be a mesh.

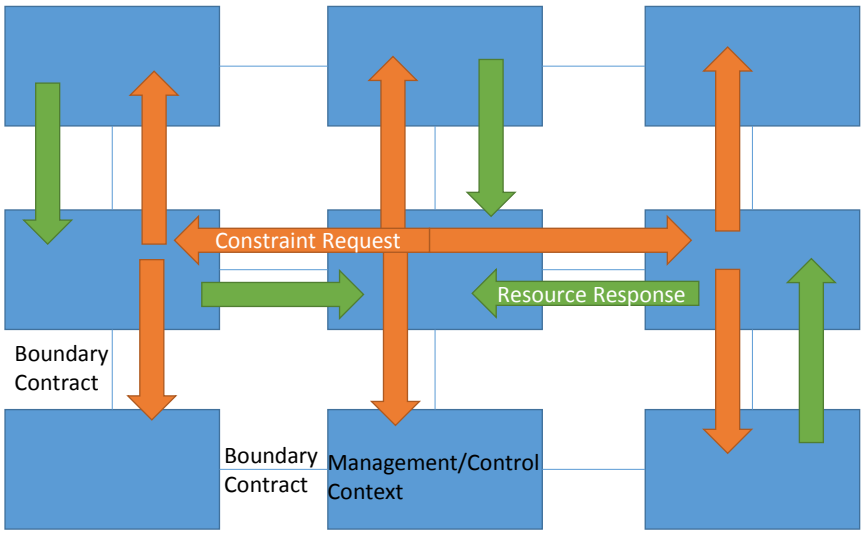


Figure 3-16 – Management/Control mesh

© 2018 Open Networking Foundation

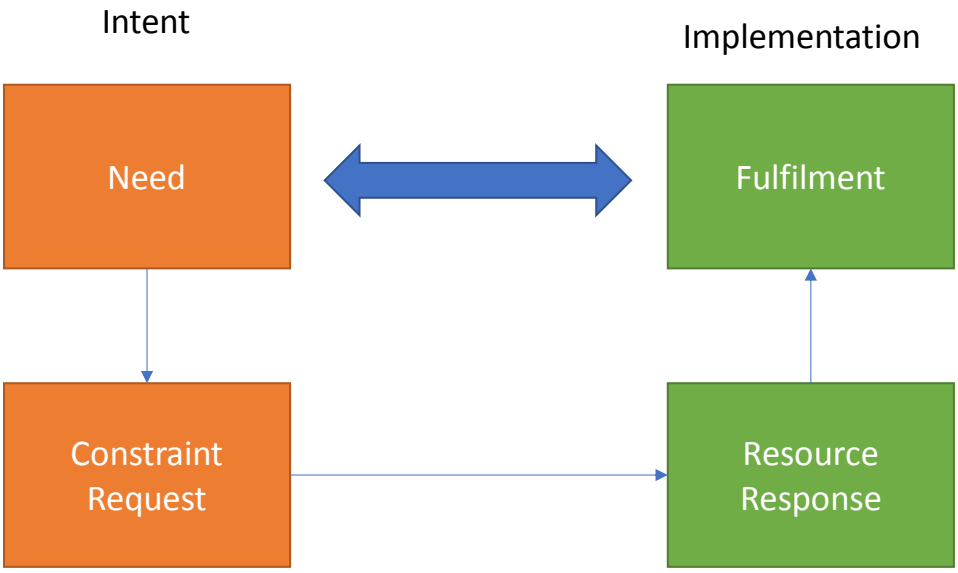


Figure 3-17 - Needs

© 2018 Open Networking Foundation

These can all be linked together into the ONF CIM model framework as shown below.

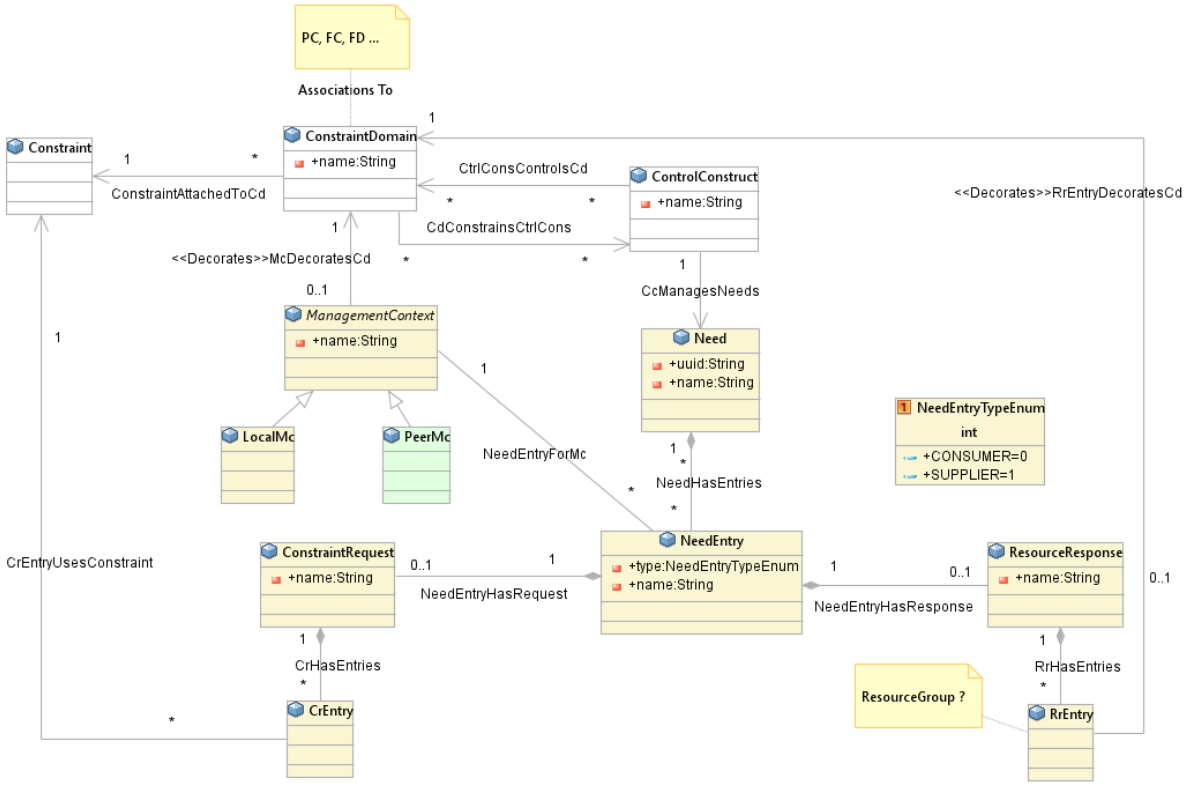


Figure 3-18 - Generalized Control Function Model

The best way of showing how the model can be used is through an example.

We will assume that our focus is on MC2 and it has neighbors MC1, 3 and 4.

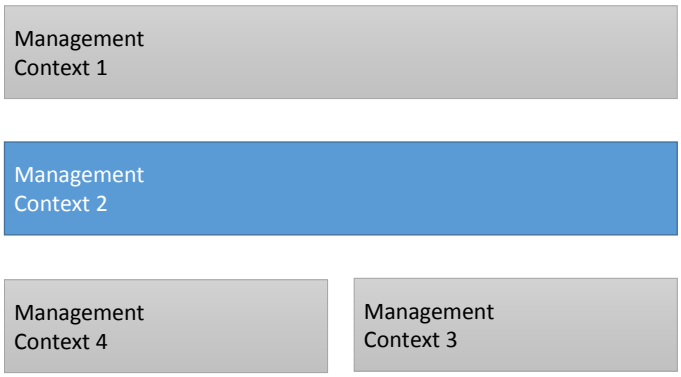


Figure 3-19 - MC Example Context

© 2018 Open Networking Foundation

The figure above simply shows the context landscape. The figure below provides some detail on the interaction opportunities. In the figure MC1 and MC2 are specifically peers, i.e., on some occasions and in some respects MC1 can be superior to MC2 and on some occasions MC2 can be superior to MC1. MC3 and 4 may be peers with MC1 or subordinates or superiors.

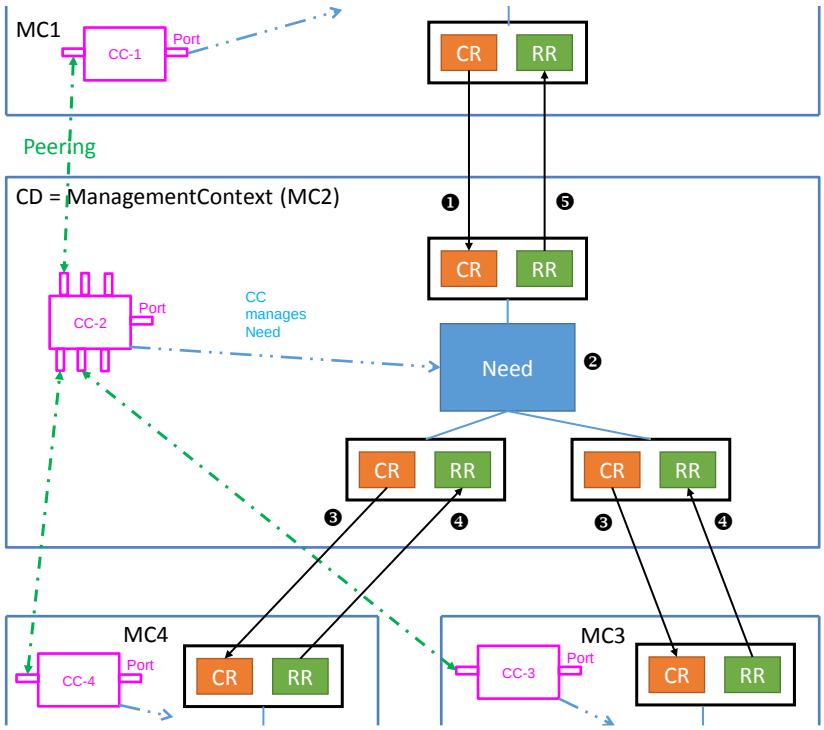


Figure 3-20 – MC Example Detail

CC-2 receives a ConstraintRequest from CC-1 (Note that in the diagram above the messages are just shown as being directly related, as drawing them all passing through the ControlConstructs would be too difficult to draw).

CC-2 creates a Need and attaches the request to it.

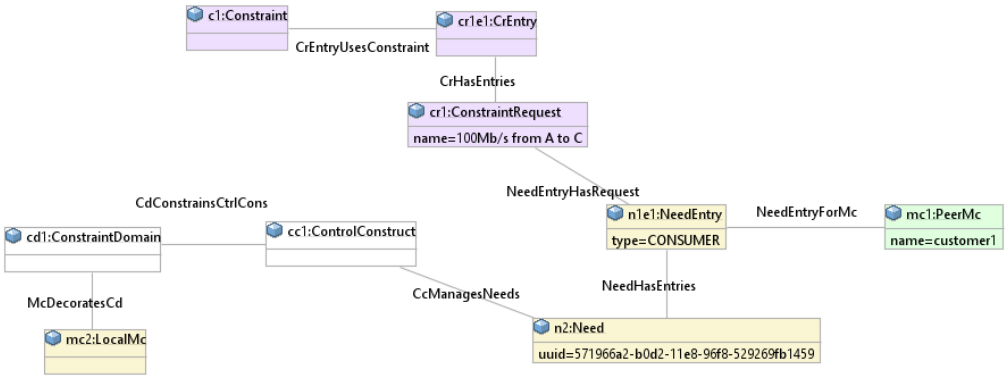


Figure 3-21 - MC Example Step 1

© 2018 Open Networking Foundation

CC-2 determines that it cannot fulfil the request itself, but that CC-3 and CC-4 can each help with part of the request, so it creates two outgoing CR and attaches these to the Need.

If CC-2 doesn't know which ControlConstruct to send the requests to then it could just broadcast the requests and then see what responses it receives (if any).

In an ideal environment, each ControlConstruct would publish a full machine-readable specification of its capability/needs. All potential relationships can be determined from this capability/need information. The specific relationships may be fleeting or long-lived depending upon the degree of engineering etc. The published capability may change over time as the policy etc. in force for a specific ControlConstruct are changed. In some cases, a ControlConstruct my only be able to be a provider or only be able to be a client, this will be conveyed via the capability information.

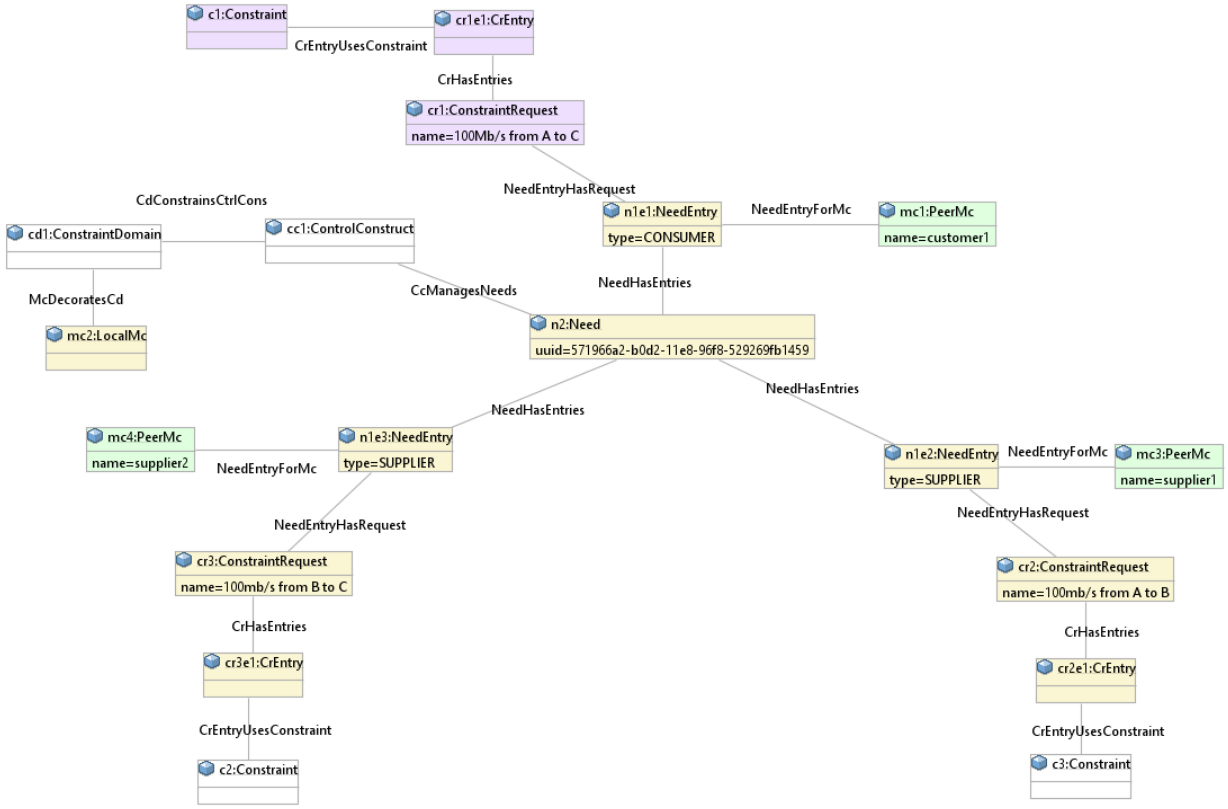


Figure 3-22 - MC Example Step 2

CC-3 and CC-4 reply with ResourceResponses that match the request they were sent. Note that we don't know if they actually fulfilled it themselves or passed it onwards again (and we don't really need to know).

CC-2 attaches these responses to the corresponding NeedEntry forming request – response pairs.

© 2018 Open Networking Foundation

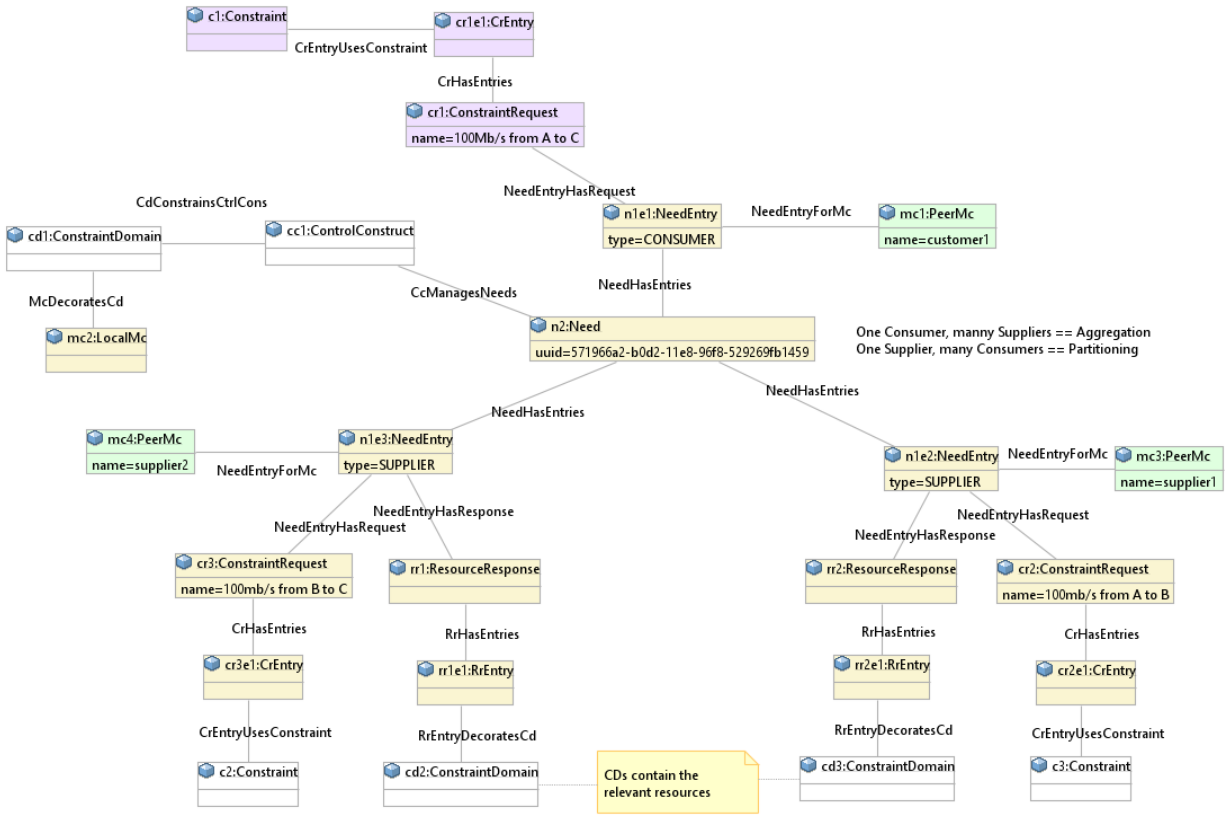


Figure 3-23 - MC Example Step 3

CC-2 can now check that it has all the resources needed to fulfil the original request.

It now creates a ResourceResponse and attaches it to the original NeedEntry and sends a response back to CC-1.

 © 2018 Open Networking Foundation

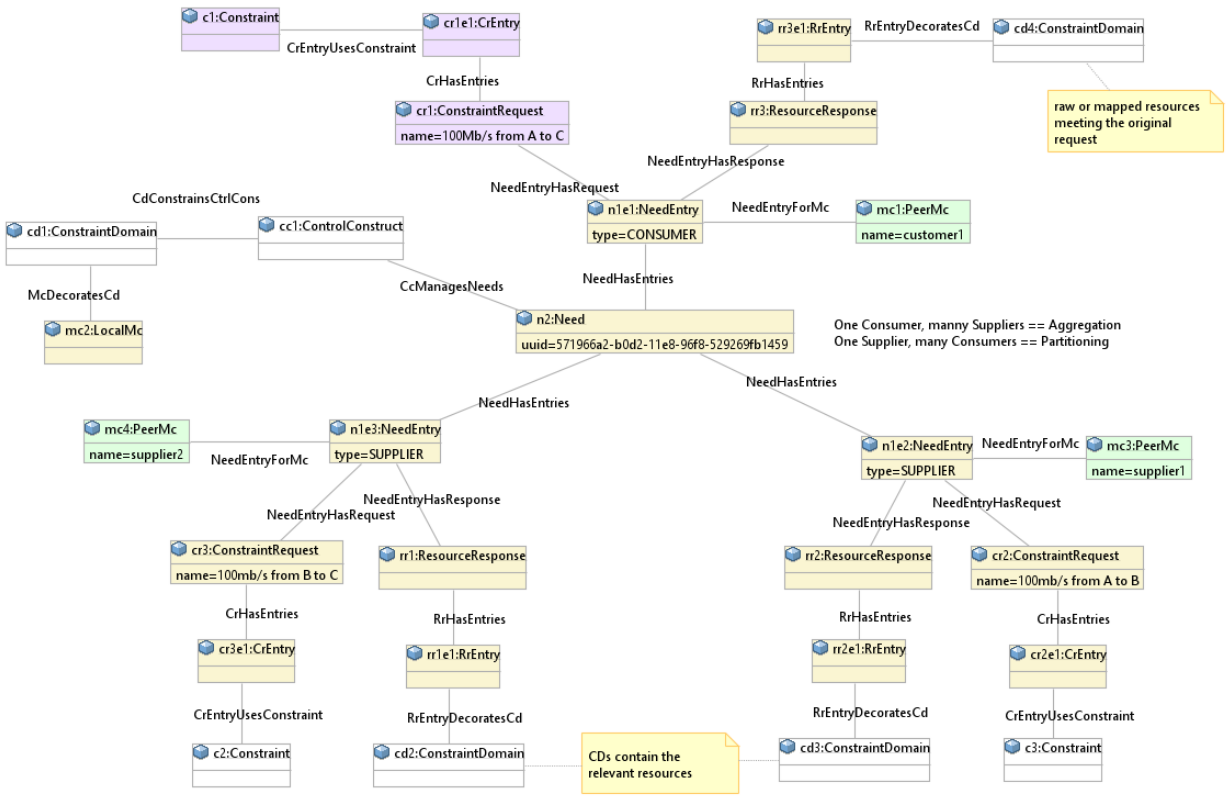


Figure 3-24 - MC Example Step 4

A similar set of steps could be used to tear down the allocations when a need is no longer required.

Note that CC-1 may not be the 'final' CC and CC-1 may have partitioned the 'original' request before sending it to CC-2.

Also it is possible that CC-1 had broadcast the request to a number of ControlConstructs and will choose between the response from CC-2 and the other responses it received.

An identifier is needed to make sure that the requests don't loop around and that a ControlConstruct can identify if one of its outward requests has been received as an inward request, which it can then ignore or reject.

In some highly engineered cases the specific relationships between ControlConstructs, and their respective roles, may be defined and known. In other more fluid cases the relationships and roles may change in very dynamic fashion.

As can be seen from this simple example, the intention is to support ControlConstructs being able to negotiate with each other to fulfil a customer request in a way that requires minimal or no manual intervention.

The result is also consistent with the management of newer architectures such as microservices and a 'service mesh'.

 © 2018 Open Networking Foundation

Having looked at how a single ControlConstruct can interact with its neighbors, we now need to look at how they can be combined into an overall control network.

Rather than having a hierarchy formed of individual ControlConstructs, it seems to make more sense to form them into layers.

The ControlConstructs would have peer control relationships within a layer and use master-slave or client-server control relationships between layers. The ControlConstructs would also have a master-slave or client-server relationship any network devices that they are controlling.

For high availability needs, ControlConstructs could be pooled (using ConstraintDomains to represent the pool groupings).

The control architecture:

- Should allow, but not impose a hierarchy on the control structure
- Should support peering of ControlConstructs and ControlConstructs groups, rather than requiring a 'super-controller' to be added, allowing:
    - Enterprise peering between regions
    - Peering between Service Providers

The following two diagrams show how this could be applied.

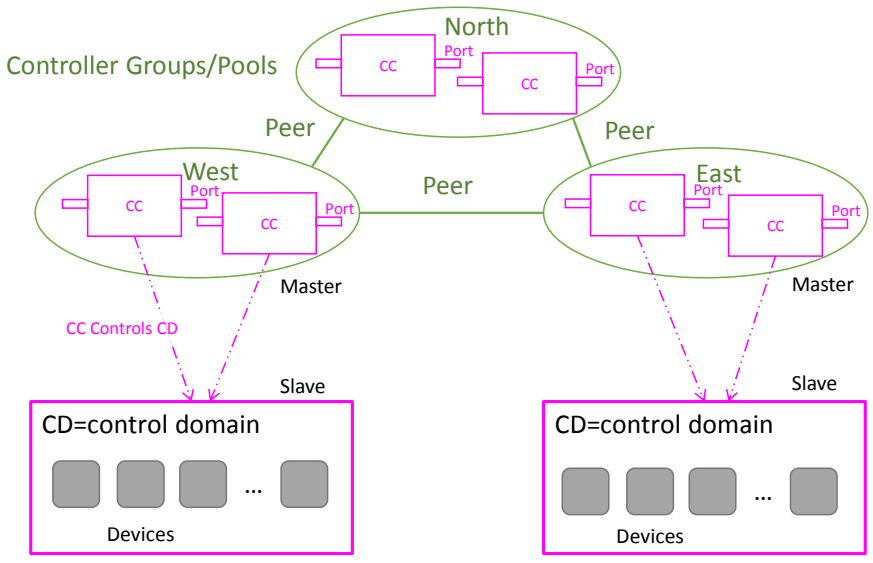


Figure 3-25 - A mix of Master-Slave and Peering

 © 2018 Open Networking Foundation

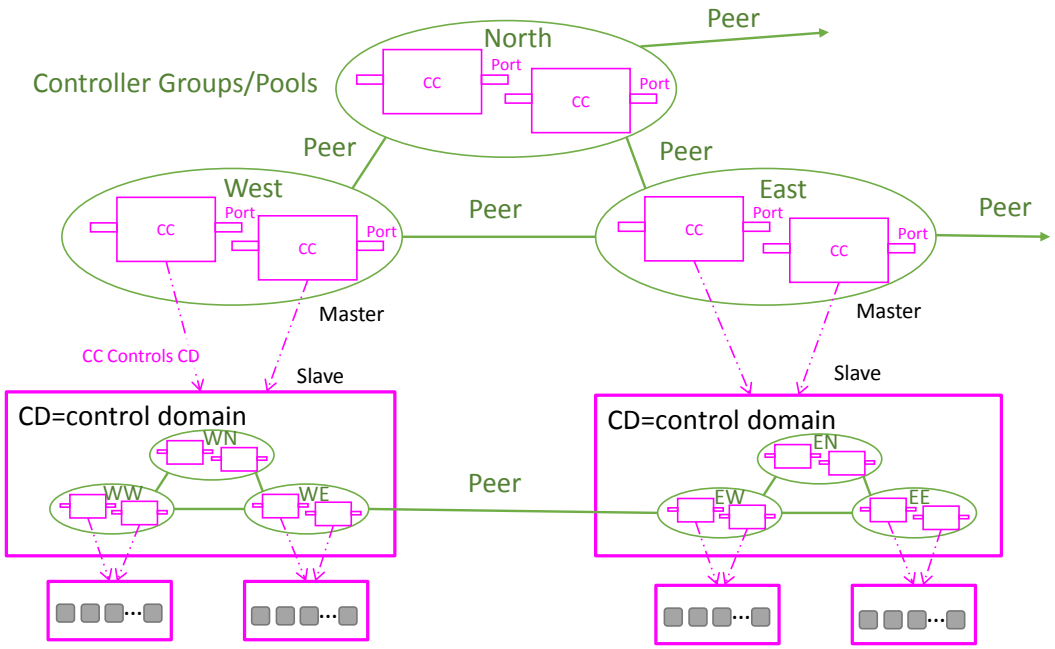


Figure 3-26 - Recursive Control Architecture

The preceding model doesn't cover how a remote ControlConstruct will access the resources assigned to it.

An "ExposureSession" concept could be defined to create a proxy with ports and addresses that can be used to control and restrict access to the resources.

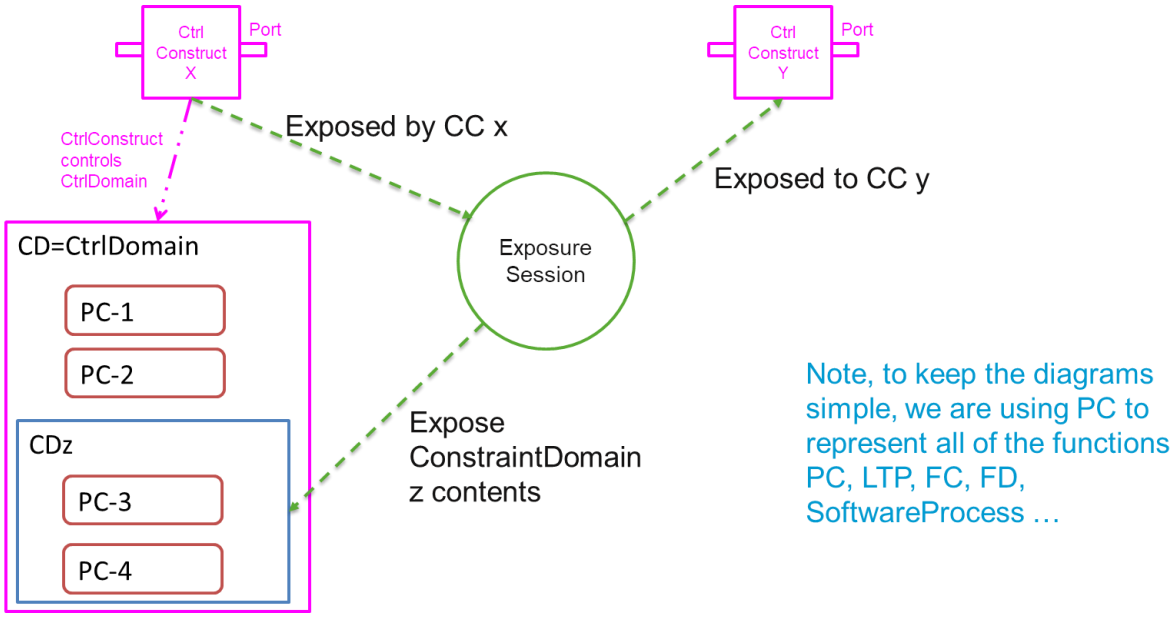


Figure 3-27 - Exposure Session allows a ControlConstruct to expose network functions to another ControlConstruct

 © 2018 Open Networking Foundation

3.4 Ethernet Ring Protection System (ERPS, ITU-T G.8032)

One last example that is worth considering is that a ControlConstruct may control a 'network' scope.

Consider ERPS G.8032 (hereafter just called ERPS).

Each Ethernet switch may be performing other normal switching functions as well as an ERPS node function. A ControlConstruct can be created for every ERPS node that receives and processes control information from other nodes.

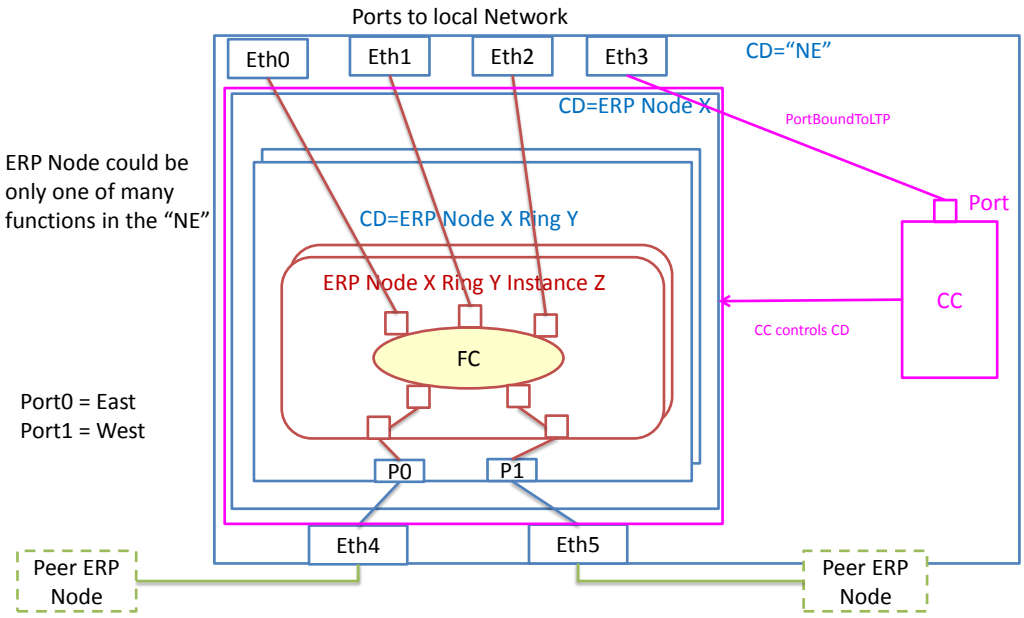


Figure 3-28 - ERP G.8032 Concept Example

We could also consider that this distributed control also creates a logical network level ControlConstruct. Once again, the building block nature of the model also allows for this case to be represented in a sensible manner.

© 2018 Open Networking Foundation

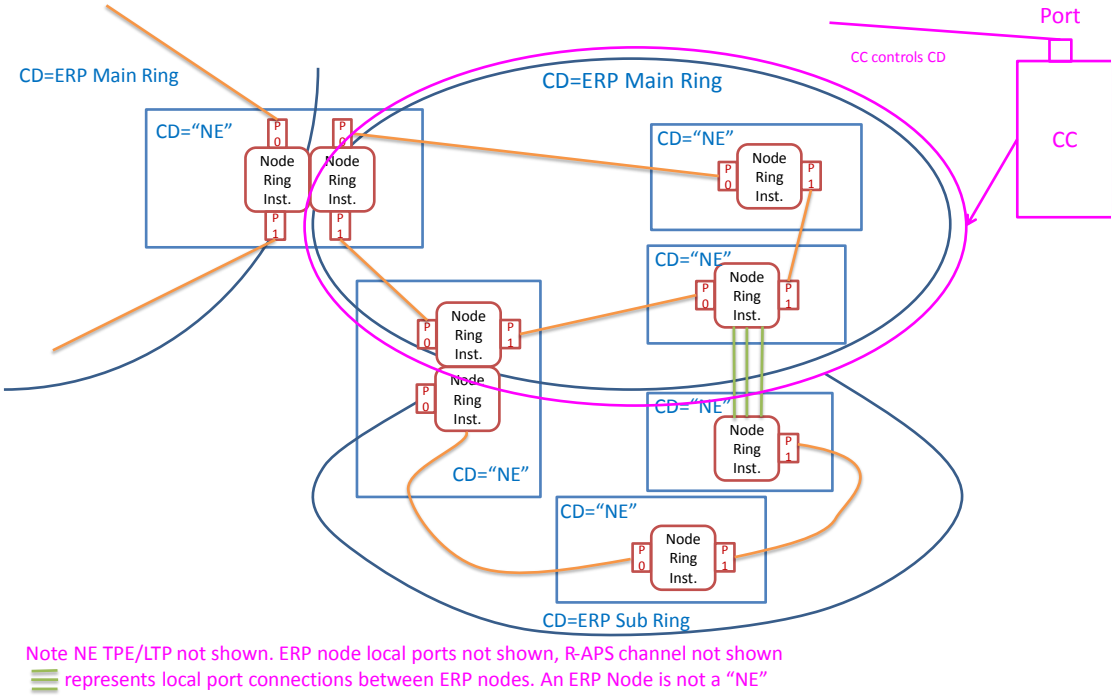


Figure 3-29 - ERP Network Example 1

End of Document

© 2018 Open Networking Foundation