<config>  
<output path=’C:\Users\ndavis\git\OnfInfoModelOutput\ModelDescriptions\TR-512.11\_OnfCoreIm-ProcessingConstruct.docx' />  
</config>

<context model=’C:\Users\ndavis\git\ONFInfoModel\OnfModel\CoreModel.uml' element=’{0}’ importedBundles='gmf;papyrus' searchMetamodels='true'/>

<gendoc><drop/>

Change path substrings above from “{path for output files}\” to your local path for the output files and “{path for CoreModel}\” to your local path for the Core Model. <drop/>

DELETE: Prior to publishing this –gd.docx (including for review), change path substrings above from “C:\Users\ndavis\git\OnfInfoModelOutput\” to “{path for output files}\” and from “C:\Users\ndavis\git\ONFInfoModel\OnfModel\” to “{path for CoreModel}\” <drop/>



Core Information Model (CoreModel)

TR-512.11

Processing Construct

Version 1.6

January 2024

ONF Document Type: Technical Recommendation

ONF Document Name: Core Information Model version 1.6

**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](http://www.opennetworking.org)

©2024 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.

Finalizing this document once generated… delete this text prior to publication:

* Replace “{{..}}” with square brackets (which trip up Gendoc)
* Select text in document from beginning of table of contents (first line) to end of document
  + Click menu item “Update Field” (on this large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
  + Repeat “update fields” 2 more times (on the same large block of text)
    - if “Update Table…” dialogue appears select “Update entire table”
* Remove reviewer comment

Note that the table of contents and figures need to be updated several times as the table length changes the page numbering and the cross references will need to be re-updated.

Table of Contents

[Disclaimer 2](#_Toc434403079)

[Open Networking Foundation 2](#_Toc434403080)

[Document History 5](#_Toc434403081)

[1 Introduction 5](#_Toc434403082)

[2 References 6](#_Toc434403083)

[3 Definitions 8](#_Toc434403084)

[3.1 Terms defined elsewhere 8](#_Toc434403085)

[3.2 Terms defined in this TR 8](#_Toc434403086)

[4 Abbreviations and acronyms 8](#_Toc434403087)

[5 Conventions 10](#_Toc434403088)

[5.1 UML modeling conventions 10](#_Toc434403089)

[5.2 Lifecycle Stereotypes 10](#_Toc434403090)

[5.3 Diagram Keys 11](#_Toc434403091)

[6 Overview of the CoreModel Fragment 13](#_Toc434403092)

[6.1 Overview of the CoreNetworkModule of the CoreModel 15](#_Toc434403093)

[6.1.1 LogicalTerminationPoint (LTP) and LayerProtocol (LP) 17](#_Toc434403094)

[6.1.2 ForwardingDomain (FD) 18](#_Toc434403095)

[6.1.3 ForwardingConstruct (FC), EndPoint (EP), FcRoute and FcSwitch 18](#_Toc434403096)

[6.1.4 Link and LinkEnd 18](#_Toc434403097)

[6.1.5 NetworkElement, NetworkControlDomain and SdnController 19](#_Toc434403098)

[6.2 CoreFoundationModule 19](#_Toc434403099)

[6.2.1 Naming and identifiers 19](#_Toc434403100)

[6.2.2 States 22](#_Toc434403101)

[6.2.2.1 Relationship between states in Provider context 22](#_Toc434403102)

[6.2.2.2 Relationship between states in the client and provider context 23](#_Toc434403103)

[6.3 Termination Fragment 24](#_Toc434403104)

[6.4 Forwarding Fragment 27](#_Toc434403105)

[6.4.1 Basic Forwarding 27](#_Toc434403106)

[6.4.2 Forwarding Construct Specification and other details of Forwarding 28](#_Toc434403107)

[6.5 Topology Fragment 32](#_Toc434403108)

[6.5.1 Basic Topology 35](#_Toc434403109)

[6.5.2 Topology and views 40](#_Toc434403110)

[6.5.3 View boundaries and intermediates 46](#_Toc434403111)

[6.5.4 More on views and names/identifiers 47](#_Toc434403112)

[6.5.5 Off-network reference and the clients view 50](#_Toc434403113)

[6.5.6 Physical Port reference 51](#_Toc434403114)

[6.5.7 Detailed properties of Topology 53](#_Toc434403115)

[6.6 Directionality 55](#_Toc434403116)

[7 Future CoreModel areas 60](#_Toc434403117)

[8 UML model files 60](#_Toc434403118)

[8.1 Papyrus File 60](#_Toc434403119)

[9 Data Dictionary 62](#_Toc434403120)

[9.1 Core Network Module data dictionary 62](#_Toc434403121)

[9.1.1 Classes 62](#_Toc434403122)

[9.1.1.1 {{cl.name/}} 62](#_Toc434403123)

[9.1.2 Data Types 65](#_Toc434403124)

[9.1.2.1 {{dt.name/}} 65](#_Toc434403125)

[9.1.3 Enumeration Types 67](#_Toc434403126)

[9.1.3.1 {{dt.name/}} 67](#_Toc434403127)

[9.1.4 Primitive Types 68](#_Toc434403128)

[9.1.4.1 {{dt.name/}} 68](#_Toc434403129)

[9.2 Core Foundation Module data dictionary 68](#_Toc434403130)

[9.2.1 Classes 68](#_Toc434403131)

[9.2.1.1 {{cl.name/}} 68](#_Toc434403132)

[9.2.2 Data Types 71](#_Toc434403133)

[9.2.2.1 {{dt.name/}} 71](#_Toc434403134)

[9.2.3 Enumeration Types 73](#_Toc434403135)

[9.2.3.1 {{dt.name/}} 73](#_Toc434403136)

[9.2.4 Primitive Types 74](#_Toc434403137)

[9.2.4.1 {{dt.name/}} 74](#_Toc434403138)

[9.3 Core Enhancements data dictionary 74](#_Toc434403139)

[9.3.1 Classes 74](#_Toc434403140)

[9.3.1.1 {{cl.name/}} 74](#_Toc434403141)

[9.3.2 Data Types 77](#_Toc434403142)

[9.3.2.1 {{dt.name/}} 77](#_Toc434403143)

[9.3.3 Enumeration Types 79](#_Toc434403144)

[9.3.3.1 {{dt.name/}} 79](#_Toc434403145)

[9.3.4 Primitive Types 80](#_Toc434403146)

[9.3.4.1 {{dt.name/}} 80](#_Toc434403147)

[9.4 Core Enhancement Fragments data dictionary 80](#_Toc434403148)

[9.4.1 Classes 80](#_Toc434403149)

[9.4.1.1 {{cl.name/}} 80](#_Toc434403150)

[9.4.2 Data Types 83](#_Toc434403151)

[9.4.2.1 {{dt.name/}} 83](#_Toc434403152)

[9.4.3 Enumeration Types 85](#_Toc434403153)

[9.4.3.1 {{dt.name/}} 85](#_Toc434403154)

[9.4.4 Primitive Types 86](#_Toc434403155)

[9.4.4.1 {{dt.name/}} 86](#_Toc434403156)

[10 Additional figures related to potential extensions 87](#_Toc434403157)

[10.1 State extensions 87](#_Toc434403158)

[10.2 LTP Specification 87](#_Toc434403159)

[10.3 Model structure rules 88](#_Toc434403160)

[11 Addendum Translation table 91](#_Toc434403161)

[12 Back matter 93](#_Toc434403162)

[12.1 Editors 93](#_Toc434403163)

[12.2 Contributors 93](#_Toc434403164)

List of Figures

[Figure 1-1 Methodology of IM and DS Development 6](#_Toc430780029)

Document History

| **Version** | **Date** | **Description of Change** |
| --- | --- | --- |
|  |  | This document was not published prior to Version 1.3 |
| 1.3 | September 2017 | Version 1.3 {{Published via wiki only}} |
| 1.3.1 | January 2018 | Addition of text related to approval status. |
| 1.4 | November 2018 | No changes. |
| 1.5 | September 2021 | Enhancements to model structure. |
| 1.6 | January 2024 | Updated release and dates. |

# Introduction to the document suite

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

## References

For a full list of references see [TR-512.1](../TR-512.1_OnfCoreIm-Overview.pdf).

## Definitions

For a full list of definition see [TR-512.1](../TR-512.1_OnfCoreIm-Overview.pdf).

## Conventions

See [TR-512.1](../TR-512.1_OnfCoreIm-Overview.pdf) for an explanation of:

* UML conventions
* Lifecycle Stereotypes
* Diagram symbol set

## 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).

## 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](../TR-512.1_OnfCoreIm-Overview.pdf) 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.

# Introduction to Processing Construct

The ProcessingConstruct (PC) represents generalized functionality. The PC is used in conjunction with the ConstraintDomain (CD), that groups PCs and constrains their usage. In addition to being generaly applicable to represent functionality that is not being modeled in detail, the PC and CD form the fundamental pattern that allows an important transition in the representation of a ‘device’[[1]](#footnote-1).

The ONF Common Information Model represents a ‘device’ with a logical functionality viewpoint utilizing an underlying physical inventory viewpoint. This decoupling of logical and functional scopes is important for us to proceed to the next stage in our representation, the removal of NetworkElement[[2]](#footnote-2).

NetworkElement (now deprecated) was the key concept that was used to represent logical functionality. In this version of the model, the new ProcessingConstruct (PC) and ConstraintDomain (CD) classes are used as a basis for the replacement of the NetworkElement concept (see [TR-512.8](TR-512.8_OnfCoreIm-Control.pdf)).

This document describes the PC and CD and provides some detail of the model.

A data dictionary that sets out the details of all classes, data types and attributes is also provided ([TR-512.D](TR-512.DD_OnfCoreIm-DataDictionary.pdf)D).

# Purpose and essentials of Processing Construct

## Background

In the ONF CIM there are already separate classes for special types of functions:

* ForwardingConstruct to represent forwarding functionality
* LogicalTerminationPoint to represent termination, and
* ForwardingDomain to represent forwarding scope constraints.

ProcessingConstruct is in addition to these concepts and is to be used where the major function of interest is related to processing rather than forwarding of information.

While there are a number of grey areas between processing and forwarding, there are a few ‘pure’ processing constructs:

* Memory
* CPU
* Storage

Another use for ProcessingConstruct is for representing control plane processes such as packet routing processes. Packet routers commonly run many routing protocols and may also run many instances of each routing protocol. Each routing process instance peers independently and using ProcessingConstruct we can show the actual control plane topologies.

While it’s hard to give extensive rules where ProcessingConstruct should be used, probably the best advice is to use it where significant processing of information takes place. Also it may be useful to “Think of the network as composed of PC, constrained by CD and connected via FC”.

ConstraintDomain has been introduced to add a ‘container’ concept. ConstraintDomain is similar to ForwardingDomain, but more general. Note that ConstraintDomain is designed as a lightweight scope boundary only, the functionality resides in PC, FC and LTP.

A useful way to think of the change is that the NetworkElement was like a black box that obscured what was happening inside. ConstraintDomain is like a light, clear wrapper around the functions, grouping and constraining them, but not adding or obscuring anything.

ConstraintDomain represents a general functional scope that can be used for grouping and scope boundaries. ConstraintDomains can be related, they can overlap and they don’t need to form a hierarchy.

ProcessingConstruct is likely to be extremely useful in the “Application Layer” which tends to be more about processing and less about forwarding.

## Model

In a number of places in the ONF CIM, the Component-Port pattern is used (e.g. ForwardingDomain & FdPort, ForwardingConstruct & FcPort). ConstraintDomain is a more generic version of ForwardingDomain and has a related CdPort class to provide at least equivalent capability. ProcessingConstruct is similar to ForwardingConstruct and has a related PcPort class to provide necessary capability for the many asymmetric cases.

In summary, we have added ConstraintDomain and ProcessingConstruct to the ONF CIM using our standard patterns. The diagram below shows the core model and emphasizes the pattern usage.

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘ProcessingConstruct-Core’, ’Processing Construct and Constraint Domain core model’)/]

[/for]<drop/>

The figure below shows the relationship between the PC/CD model and the Equipment model. The figure highlights an enhancement in the relationship between equipment and the function model (covered in detail in [TR-512.6](TR-512.6_OnfCoreIm-Physical.pdf)). The model emphasizes that PC can represent anything from the most fundamental small (atomic) functions to the richest and most sophisticated of functions. The recursion shown allows the equipment to give rise to a block of functionality that can be decomposed into atomic parts and then those parts can be reassembled, perhaps through a number of intermediate forms to eventually give rise to LTPs etc. In addition, at any level of assembly, the functionality can be made resilient by selecting from two or more alternatives[[3]](#footnote-3).

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘ProcessingConstruct-EquipmentAndLtpWithCore’, ’PC and CD with Equipment and LTP’)/]

[/for]<drop/>

In the figure below other model classes have been added. The figure emphasizes the similarity between the patterns of the CD/PC model and the FD/FC model. The FD/FC is essentially a narrow usage of CD/PC.

In the model (as shown in the figure below):

* One or more ConstraintDomains can constrain ProcessingConstruct via the CdConstrainsPc association (this is a more general capability than that of ForwardingDomain constrains ForwardingConstruct).
* Two or more ConstraintDomains can be related together via the CdEncompassesCd association (in the same way the ForwardingDomain can).
* One or more ConstraintDomains can constrain ForwardingConstruct via the CdConstrainsFc association.
* PcPort is related to LogicalTerminationPoint in the same way that FcPort is.
* ConstraintDomain can constrain ForwardingDomains (note that ConstraintDomain and ForwardingDomain are very similar concepts)[[4]](#footnote-4)
* ConstraintDomain association to Equipment allows us to relate the constraint boundary to an underlying physical entity.
* ForwardingDomain can constrain ProcessingConstruct (in the same way that ForwardingDomain constrains ForwardingConstruct)[[5]](#footnote-5).
* The association “EquipmentGivesRiseToFunctionalBlock” applies to a low level function that is wholly “contained” by an FRU[[6]](#footnote-6).

[for(p:Package|Package.allInstances())]<drop/>

Inserts the diagram identified in first quotes with the title identified in second quotes <drop/>  
[p.insertStandardDiagram(‘ProcessingConstruct-FullModel’, ’PC and CD with FD, FC and C&SC’)/]

[/for]<drop/>

The sections below provide details of the classes and relationships.

Note that not all attributes/details are shown for the classes below (see [TR-512.DD](TR-512.DD_OnfCoreIm-DataDictionary.pdf) for a list of all attributes). Only those attributes that are relevant for this document are shown.

## PC/CD model

### ProcessingConstruct (PC)

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘ProcessingConstructModel’))]<drop/>

[if (cl.name.contains(‘ProcessingConstruct’))]<drop/>

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’ProcessingConstructModel’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### PcPort

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘ProcessingConstructModel’))]<drop/>

[if (cl.name.contains(‘PcPort’))]<drop/>

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’ProcessingConstructModel’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### ConstraintDomain (CD)

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘ProcessingConstructModel’))]<drop/>

[if (cl.name.contains(‘ConstraintDomain’))]<drop/>

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’ProcessingConstructModel’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

### CdPort

[for (cl:Class | Class.allInstances()->sortedBy(name))]<drop/>

[if (cl.qualifiedName.contains(‘ProcessingConstructModel’))]<drop/>

[if (cl.name.contains(‘CdPort’))]<drop/>

Inserts the details of the class in first quotes from the package in second quotes <drop/>  
[cl.insertClass(cl.name,’ProcessingConstructModel’)/]

Inserts the attributes of the class <drop/>   
[cl.insertAttributeTableBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

## Related Classes

In version 1.3.1 the FD, FC and Equipment were listed as classes impacted by the addition of PC/CD. In version 1.3.1 FD, FC and Equipment all referred to PC and/or CD. In version 1.4 these references have been removed. This reduces model coupling with no loss of capability. References, where relevant, are now from PC/CD to other classes in the model.

# Explanatory Figures

This section provides figures that illustrate the application of the model to various network scenarios. Further examples are provided in [TR-512.A.9](TR-512.A.9_OnfCoreIm-Appendix-ProcessingConstructExamples.pdf).

In this section, we will use some simple examples to help in understanding the intention of the ProcessingConstruct and ConstraintDomain classes, but not go into the level of detail that we will in the examples document.

## Multi- Functional Device Example

As stated in the introduction, moving from a traditional Network Element oriented model to ProcessingConstruct oriented model requires us to think differently about a ‘device’. Rather than thinking about a large functional block that can be ‘decomposed’, we should now be thinking of smaller functions that can be ‘assembled’.

In general, a ‘device’ instance is represented by one or more ControlComponent (representing a relevant part of the control functionality of the device that are accessible through a ControlPort – see [TR-512.8](TR-512.8_OnfCoreIm-Control.pdf)), one or more ControlSystemViews (where each represents a grouping of information about the device (in the form of object instance) available as a view – see [TR-512.8](TR-512.8_OnfCoreIm-Control.pdf)) and one or more ConstraintDomain (where each ConstraintDomain represents a set of interrelated elements of the device) along with instances of PC, LTP, FD, FC, Equipment etc to represent various functional and physical arrangements.

In the diagram below, we show a ‘device’ where we have deliberately removed most of the details. The NetworkElement scope (which we assume was based on a ‘chassis’ physical scope in this case) has been replaced by a ControlComponent , a ControlSystemView and a ConstraintDomain. The ControlComponent and ControlSystemView are described in [TR-512.8](TR-512.8_OnfCoreIm-Control.pdf)).

This particular ‘device’ is actively providing four functions (and a real case could have many more). Each of these functions is peering with similar functions in other ‘devices’ forming protocol topologies.

For the BGP control plane function we show that we can relate BGP peers by directly binding the PC ports (using PcPortBoundToPcPort) to give a simplified application view and we can also represent the transport viewpoint (via LTP, FD, FC, Link …).



Figure 6-2 Using PC to represent complex functions and CD to constrain

## Distributed Device

Because this is the overview document, we won’t go through this example in detail, but it shows the generality of the ConstraintDomain concept.



Figure 6-2 Distributed Device – Split Chassis

In the example, there are 3 chassis, with a central chassis and two satellite chassis connected. The central chassis controls only part of each satellite chassis.

At the bottom of the Logical View there is a ConstraintDomain related to each chassis, which enables enforcement of physical inventory related constraints.

For each satellite, there is a ConstraintDomain to represent the logical split in the chassis.

So, the distributed device can be viewed as a whole, there is also a Constraint Domain shown as ‘Aggregated CD’.

There may also be some ProcessingConstructs that cross the chassis boundary (such as Ethernet switching) – it’s just a matter of determining which constraint scope is appropriate.

Similarly, in the management plane, there may be different ControlSystemViews with different management-control scope / visibility.

## PTP Clock Example

For this example, we will assume that:

* A device may have zero, one or more PTP clocks
* For each clock:
  + Clock input may be received from 1 or more foreign masters on (incoming) slave ports
  + An algorithm is used to select one of the clock inputs to use
  + This selected clock input can be propagated to zero, one or more clocks via (outgoing) master ports
  + Only ports enabled for PTP can have PTP packets extracted or injected.

Note that a port may see more than one foreign master (e.g. if there is a bridge in-between).

Note that messages are sent in the reverse direction too, this diagram just shows the clock Master – Slave relationships

This enables us to produce a simple functional view of a PTP Clock ProcessingConstruct in a ‘device’, as shown in the diagram below.



Figure 6-2 PTP Clock Concepts

We can see that the ‘device’ boundary is represented by a ConstraintDomain with related LTP.

Because of the PTP clock asymmetry, we need to create PcPorts and assign roles to them.

The Ethx LTP are bound to the PcPort (via association PcPortBoundToLTP).

We can also show the direct PTP topology by directly binding the PC ports (using PcPortBoundToPcPort) as shown by the purple dotted line.

Note that we are using the term ‘PTP Clock’ to avoid confusion of ‘Precision Time Protocol’ with ‘Physical Termination Point’.

## ERPS G.8032 Example.

Assume an Ethernet switch that can support ERPS (Ethernet Ring Protection Switching) functionality (in addition to its other functions). As before, there is a ConstraintDomain that relates to the physical boundary and related LTP.

Assume the ‘device’ supports a single ERPS node only, hence there is a single ConstraintDomain inside the ‘device’ representing the ERPS Node. An ERPS node can participate in many rings and each ring can have many instances. A ConstraintDomain represents each ring and a ProcessingConstruct represents each ring instance, nested as shown below.



Figure 6-2 ERP {{ITU-T G.8032}} Concept Example

Note how the ConstraintDomain ring ports Port0 and Port1 are bound to the ‘device’ LTP.

Also note that inside of each ring instance in the node, there is a ForwardingConstruct that shows the ring flows.

ConstraintDomains can be used at the network level, as shown in the diagram below, to highlight ring scope.

Note also that the ‘device’ level constraint domains cross the ring ConstraintDomain boundaries, so this is an example where constraint domains don’t form a hierarchy. This supports a ‘ring viewpoint’ as well as a ‘device viewpoint’ and shows that certain constraints are ring or ‘device’ scoped.

In the [TR-512.5](TR-512.5_OnfCoreIm-Resilience.pdf) the ConfigurationAndSwitchController is shown in place of the generalized PC as at this stage a specialist Resilience controller is use for all resilience cases.



Figure 6-2 ERP Network Example

# Further considerations (see also [TR-512.FE](TR-512.FE_OnfCoreIm-FutureEnhancements.pdf))

1. Degree of specialization:

* PC v ControlComponent v C&SC, PC v FC – should all functional components be PC rather than specialized?
* CD v FD – should all domains be CD rather than specialized?

1. Further detailing the PC model:

* Dealing with the component – system recursion
* Projecting views

1. Modeling software to provide more precise representation of emergent PC (currently only equipment supports PC).
2. CD/PC spec and relationship to the generalized spec approach

**End of document**

</gendoc><drop/>

To take latest template: <drop/>

* delete text from “Template version…” to end of file <drop/>
* insert a line in “Normal” style<drop/>
* insert text (Insert 🡪 Object 🡪 Text from File… (alt njf)) from: <drop/>
  + TR-512.GT\_OnfCoreIm-CommonGendocTemplate-Fragments.docx <drop/>

Template version 0.0.10 17 September 2017 <drop/>

# Fragment: Insert class <drop/>

<fragment name=’insertClass’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
<arg name=’className’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (not cl.qualifiedName.contains(packageName))]<drop/>  
[else] <drop/>  
[if(cl.name.contains(className))]<drop/>

Qualified Name: [cl.qualifiedName/]

[for (co:Comment | cl.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (cl.isAbstract)]<drop/>

This class is abstract.

[/if]<drop/>

[if (cl.oclAsType(uml::Class).general ->notEmpty())]<drop/>

Inherits properties from:

[for (gen:Class | cl.oclAsType(uml::Class).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | cl.getAppliedStereotypes())]<drop/>  
[if(not st.name.contains(‘OpenModelClass’))]<drop/>

This class is [st.name/].

[else] <drop/>  
[/if]<drop/>  
[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Insert standard diagram <drop/>

<fragment name=’insertStandardDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 6-2 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert small diagram <drop/>

<fragment name=’insertSmallDiagram’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Package’/><drop/>  
<arg name=’diagramName’ type=’String’/><drop/>  
<arg name=’diagramTitle’ type=’String’/><drop/>

[for (d:Diagram|p.getPapyrusDiagrams())]<drop/>

[if d.name.contains(diagramName)]

<drop/>

<image object='[d.getDiagram()/]' maxW='true' keepH='false' keepW = ‘false’></image>

CoreModel diagram: [d.name/]

Figure 6-2 [diagramTitle/]

[else]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief not Obsolete<drop/>

<fragment name=’insertAttributeRowBriefNotObsolete’ importedBundles=’commons;gmf;papyrus’><drop/>

Does not work unless we have Mature stereotype… <drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

[for (st:Stereotype | p.getAppliedStereotypes())]<drop/>

[if(not st.name.contains(‘OpenModelAttribute’))]

[if(not st.name.contains(‘Obsolete’))]

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>  
</fragment><drop/>

# Fragment: Insert attribute row brief <drop/>

<fragment name=’insertAttributeRowBrief’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’p’ type=’uml::Property’/><drop/>

| [p.name/] | [for (st:Stereotype | p.getAppliedStereotypes())]<drop/>  [if(not st.name.contains(‘OpenModelAttribute’))] [st.name/]  [/if]<drop/>  [/for]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> | [if p.ownedComment->notEmpty()]<drop/>  [for (c:Comment | p.ownedComment)] <drop/>  [cleanAndFormat(c.\_body.clean())/]  [/for]  [else] [if (p.name.contains (‘\_’))]See referenced class  [else]To be provided  [/if]<drop/>  [/if]<drop/>  Do NOT remove the previous line as word throws an error if the cell is empty <drop/> |
| --- | --- | --- |

</fragment><drop/>

# Fragment: Start attribute table brief <drop/>

<fragment name=’insertAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Attribute table brief <drop/>

<fragment name=’insertAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert Ten Specified Attribute table brief <drop/>

<fragment name=’insertTenSpecifiedAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’cl’ type=’uml::Class’/><drop/>

<arg name=’p1’ type=‘String’/><drop/>

<arg name=’p2’ type=‘String’/><drop/>  
<arg name=’p3’ type=‘String’/><drop/>  
<arg name=’p4’ type=‘String’/><drop/>  
<arg name=’p5’ type=‘String’/><drop/>  
<arg name=’p6’ type=‘String’/><drop/>  
<arg name=’p7’ type=‘String’/><drop/>  
<arg name=’p8’ type=‘String’/><drop/>  
<arg name=’p9’ type=‘String’/><drop/>  
<arg name=’p10’ type=‘String’/><drop/>  
[if cl.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [cl.name/]

<table><drop/>

[cl.insertAttributeTableHeader ()/]

[for (p:Property|cl.ownedAttribute)]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (not p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[if (p.name.contains(p1) or p.name.contains(p2) or p.name.contains(p3) or p.name.contains(p4) or p.name.contains(p5) or p.name.contains(p6) or p.name.contains(p7) or p.name.contains(p8) or p.name.contains(p9) or p.name.contains(p10))]<drop/>

[if (p.name.contains(‘\_’))]<drop/>

[p.insertAttributeRowBrief ()/]

[/if]<drop/>

[/if]<drop/>

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert DataType <drop/>

<fragment name=’insertDataType’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
<arg name=’dataTypeName’ type=’String’/><drop/>  
<arg name=’packageName’ type=’String’/><drop/>  
[if (dt.qualifiedName.contains(packageName))]<drop/>  
[if(dt.name.contains(dataTypeName))]<drop/>

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>  
[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits properties from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]<drop/>

[for (gen:Class | dt.oclAsType(uml::DataType).general)]<drop/>

* [gen.name/]

[/for]<drop/>

[/if]<drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>  
This class is [st.name/].

[/for]<drop/>  
[else] <drop/>  
[/if]  
[/if]  
</fragment><drop/>

# Fragment: Start Data Type attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableHeader’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

|  |  |  |
| --- | --- | --- |
| **Attribute Name** | **Lifecycle Stereotype (empty = Mature)** | **Description** |

</fragment><drop/>

# Fragment: Insert Data Type Attribute table brief <drop/>

<fragment name=’insertDataTypeAttributeTableBrief’ importedBundles=’commons;gmf;papyrus’ importedFragments='insertDataTypeAttributeTableHeader;insertAttributeRowBrief’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>  
[if dt.ownedAttribute->notEmpty()]<drop/>

Table 1: Attributes for [dt.name/]

<table><drop/>

[dt.insertDataTypeAttributeTableHeader ()/]

[for (p:Property|dt.ownedAttribute)]<drop/>

[p.insertAttributeRowBrief ()/]

[/for]<drop/>

</table><drop/>

[/if]<drop/>

</fragment><drop/>

# Fragment: Insert enums <drop/>

<fragment name=’insertEnums’ importedBundles=’commons;gmf;papyrus’><drop/>  
<arg name=’dt’ type=’uml::DataType’/><drop/>

#### [dt.name/]

Qualified Name: [dt.qualifiedName/]

[for (co:Comment | dt.ownedComment)]<drop/>

<dropEmpty>[cleanAndFormat(co.\_body.clean())/]</dropEmpty>

[/for]<drop/>

Applied stereotypes:

[if dt.getAppliedStereotypes()->notEmpty()] <drop/>

[for (st:Stereotype | dt.getAppliedStereotypes())]<drop/>

* [st.name/]

[/for]<drop/>

[else] No stereotypes applied

[/if]<drop/>

[if (dt.oclAsType(uml::DataType).general ->notEmpty())]<drop/>

Inherits literals from:

[for (tp:DataType | dt.oclAsType(uml::DataType).general)]<drop/>

* [tp.name/]

[/for]

[/if]<drop/>

[if (dt.oclAsType(Enumeration).ownedLiteral->notEmpty())]<drop/>

Contains Enumeration Literals:

[for (e:EnumerationLiteral|dt.oclAsType(Enumeration).ownedLiteral)]<drop/>

* [e.name/]:
  + [for (co:Comment | e.ownedComment)]<drop/>
  + <dropEmpty>[cleanAndFormat(co.\_body.clean())/]
  + </dropEmpty>[/for]<drop/>
  + [if dt.getAppliedStereotypes()->notEmpty()] <drop/>
  + Applied stereotypes:
    - [for (st:Stereotype | e.getAppliedStereotypes())]<drop/>
    - [st.name/]
    - [/for]<drop/>
  + [/if]<drop/>

[/for]<drop/>

[/if]<drop/>

</fragment><drop/>

1. Here we will use the term ‘device’ in a loose and undefined manner to aid in the discussion. The term is not defined because it is not important for our discussion, the generally understood concept is sufficient. [↑](#footnote-ref-1)
2. Deprecation of the formal concept and of the corresponding model class. [↑](#footnote-ref-2)
3. Clearly this is a basic scheme and further work is required in this area. It is likely that a level of sophistication similar to the FC resilience will be required. In addition there is a need to model load sharing and other analogue relationships which has also not been covered. [↑](#footnote-ref-3)
4. Because of the similarity of functionality between CD and FD it makes sense to review CD and FD in a future release development cycle to determine if there is an opportunity for convergence. [↑](#footnote-ref-4)
5. This and the reverse association are highlighted in red as the rationale for the associations requires further work. [↑](#footnote-ref-5)
6. There are two potential issues with this association: (1) navigation direction wrt coupling of model domains (2) essential duplication with PcRealized…association. There are some special subtle challenges highlighted here: (1) at this base level the navigation is strongly the reverse of the PcRealized… association (2) it is strongly compositional unlike PcRealized... This is a case where association navigability, strength and multiplicity vary per case. This area will be addressed in the next release. [↑](#footnote-ref-6)