Core Information Model (CoreModel)

TR-512.17 Foundation (State)

Version 1.6
January 2024

ONF Document Type: Technical Recommendation
ONF Document Name: Core Information Model version 1.6

Disclaimer

Open Networking Foundation
1000 El Camino Real, Suite 100, Menlo Park, CA 94025
www.opennetworking.org

Important note

This Technical Recommendations has been approved by the Project TST, but has not been approved by the ONF board. This Technical Recommendation is an update to a previously released TR specification, but it has been approved under the ONF publishing guidelines for 'Informational' publications that allow Project technical steering teams (TSTs) to authorize publication of Informational documents. The designation of '-info' at the end of the document ID also reflects that the project team (not the ONF board) approved this TR.

Table of Contents

List of Figures

Document History

Version	Date	Description of Change
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]
1.3	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	State aspects of TR-512.3 split out to form this document, TR-512.17.
1.6	January 2024	Updated release and dates.

1 Introduction to the document suite

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

1.1 References

For a full list of references see TR-512.1.

1.2 Definitions

For a full list of definition see TR-512.1.

1.3 Conventions

See TR-512.1 for an explanation of:

- UML conventions
- Lifecycle Stereotypes
- Diagram symbol set

1.4 Viewing UML diagrams

Some of the UML diagrams are very dense. To view them either zoom (sometimes to 400%) 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.

2 Introduction to the Foundation Model

The focus of this document is the parts of Core Foundation Model of the ONF-CIM that deal with states.

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

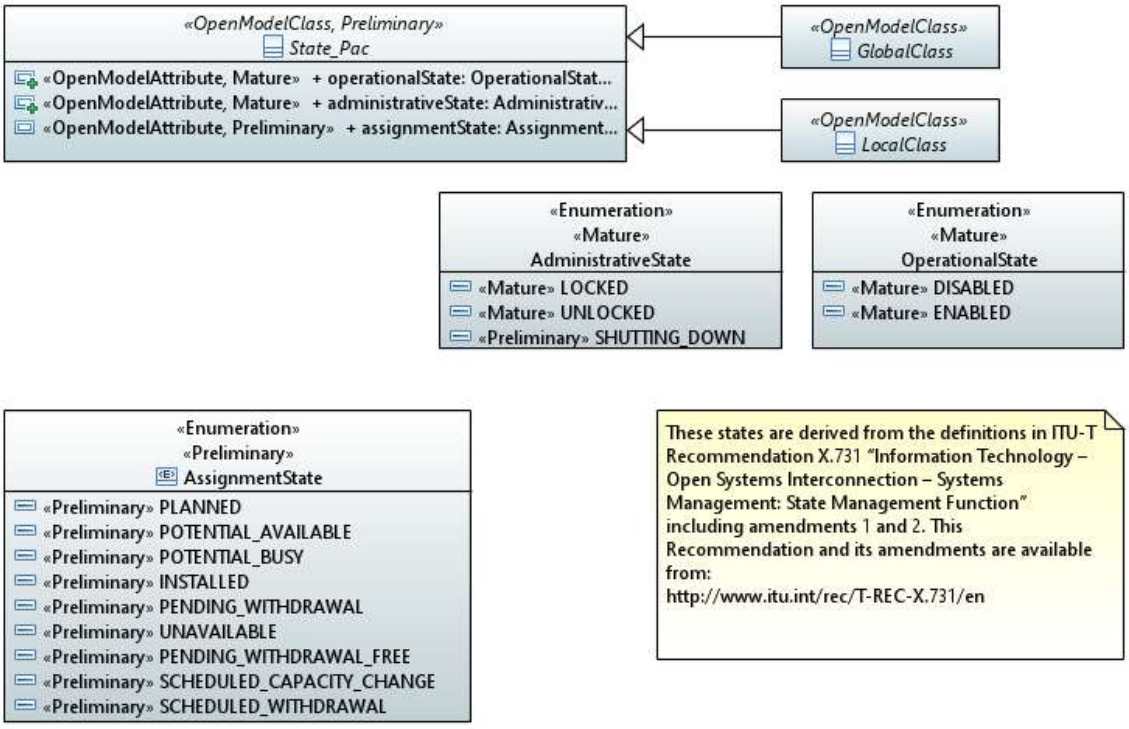
3 CoreFoundationModel

This Model includes all aspects of the core that are relevant to all other parts of the ONF CIM such as identifiers, naming and states. This document addresses the states.

3.1 States

The Core Foundation Model also defines a State_Pac artifact, which provides state attributes. The work on states is preliminary at this stage (it is derived from [ITU-T X.731]). The State_Pac is inherited by GlobalClass and LocalClass object classes. The use of these states provides a consistent way represent the overall operability, usability and current usage of the resource.

It should be noted that the states are «Mature»/«Preliminary».



CoreModel diagram: State-FullModel

Figure 3-1 States for all Objects

The states described above apply to individual abstractions, it is also important to understand how the states of an abstraction are related to the states of the supporting abstractions. The relationship between the states of the abstractions is illustrated in Figure 3-2 below.

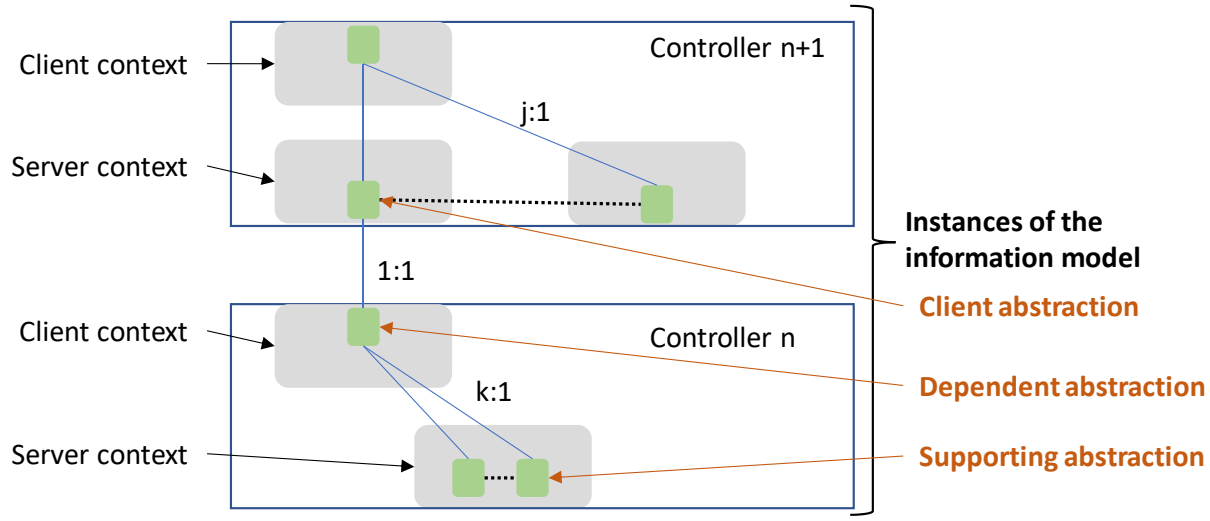


Figure 3-2 Relationship between abstractions

A description of the client abstraction, dependent abstraction and supporting abstraction is provided in section 3.1.6.

3.1.1 Classes and attributes

3.1.1.1 State_Pac

Qualified Name: CoreModel::CoreFoundationModel::StateModel::ObjectClasses::State_Pac

Provides general state attributes.

This class is abstract.

This class is Preliminary.

Table 1: Attributes for State_Pac

Attribute Name	Lifecycle Stereotype (empty = Mature)	Description
operationalState	Mature	The operational state is used to indicate whether or not the resource is installed and working.
administrativeState	Mature	Shows whether or not the client has permission to use or has a prohibition against using the resource. The administrative state expresses usage permissions for specific resources without modification to the provisioning of those resources.
assignmentState	Preliminary	Used to track the planned deployment, allocation to clients and withdrawal of resources.

3.1.2 Enumerations

3.1.2.1 AdministrativeState

Qualified Name:
CoreModel::CoreFoundationModel::StateModel::TypeDefinitions::AdministrativeState

The administrative state is used to show whether use of a resource is allowed or prohibited. The administrative state can be observed and directly controlled by certain operational roles. Typically, only a user with administrative privileges is allowed to write the administrative state, any other users are restricted to read only.

Applied stereotypes:

- Mature

Contains Enumeration Literals:

- LOCKED:
  - Users are administratively prohibited from making use of the resource.
  - Applied stereotypes:
    - Mature
- UNLOCKED:
  - Users are allowed to use the resource.
  - Applied stereotypes:
    - Mature
- SHUTTING_DOWN:
  - The resource is administratively restricted to existing instances of use only. There may be specific actions to remove existing uses. No new instances of use can be enabled.
    The resource automatically transitions to "locked" when the last user quits.
    The assignment state PENDING_WITHDRAWAL should be used to indicate to the client that the provider intends to withdraw the resource from service.
  - Applied stereotypes:
    - Preliminary

3.1.2.2 AssignmentState

Qualified Name:
CoreModel::CoreFoundationModel::StateModel::TypeDefinitions::AssignmentState

This state is used to track the planned deployment, allocation to clients and withdrawal of resources.

Applied stereotypes:

- Preliminary

Contains Enumeration Literals:

- PLANNED:

  - The resource is planned but is not present in the network or has not been made available for use.

    The following additional information may also be provided:
    - Time: An indication of when the resources are expected to be available for use.
    - Comments on the status of the plan: For example:
    --- Preliminary – Initial plan, subject to change
    --- Committed
    --- Installation in progress
    --- Client request

  - Applied stereotypes:
    - Preliminary
- POTENTIAL_AVAILABLE:
  - The supporting resources are present in the network and available for use. The resources are shared with other clients but are not currently in use.
    A temporal expression is used to indicate when the resource will allocated to the client.
    (1) When a potential resource is configured and allocated to a client it is moved to the SCHEDULED_WITHDRAWAL state for that client.
    (2) If the potential resource has been consumed (e.g., allocated to another client) it is moved to the POTENTIAL_BUSY state for all other clients.
  - Applied stereotypes:
    - Preliminary
- POTENTIAL_BUSY:
  - The supporting resources are either present in the network but are not available for use by this client or, the resources have not been installed.
    - A temporal expression is used to indicate when the resource will free (i.e., POTENTIAL_AVAILABLE) or will be allocated to the client (i.e., will be moved to SCHEDULED_WITHDRAWAL for the client).

  - Applied stereotypes:
    - Preliminary
- INSTALLED:
  - The resource is present in the network has been allocated to the client (i.e., the resource is not shared) and should be capable of providing the service.

    Note that if a resource is shared, then the SCHEDULED_WITHDRAWAL or SCHEDULED_CAPACITY_CHANGE enumeration is used (instead of INSTALLED) to indicate that the resource has been allocated to the client for a defined period of time and should be capable of providing service.
  - Applied stereotypes:
    - Preliminary
- PENDING_WITHDRAWAL:
  - The resource has been marked for withdrawal (e.g., to allow maintenance or removal of the resource). Should include

- Withdrawal Time: Indicates when the resources will be withdrawn
- Return Time: Indicates when the resources are expected to be made available for use.
If the resource will not be returned to service, then the return time is empty.

Notes:
- If the return time is empty the abstraction (including the UUID) should be deleted after the resource is withdrawn
- If a return time is defined the abstraction should be moved to the UNAVAILABLE state after the resource is withdrawn

    - Applied stereotypes:
        - Preliminary
- UNAVAILABLE:
    - The resource is present in the network but is unable to provide service for a predefined period of time (e.g., maintenance is being performed on the resource).
      Should include:
      - Time: Indicates when the resource is expected to be available for use.

    - Applied stereotypes:
        - Preliminary
- PENDING_WITHDRAWAL_FREE:
    - - Only used in a dependent abstraction or a client abstraction
      - The resource is not currently in use and the provider may be withdraw the resource
      (without causing disruption to the client service).

    - Applied stereotypes:
        - Preliminary
- SCHEDULED_CAPACITY_CHANGE:
    - The resource is present in the network. It is shared with other clients and the capacity available to the client changes over time.
      - A temporal express is used to indicate when the capacity allocated to the client will be changed.

    - Applied stereotypes:
        - Preliminary
- SCHEDULED_WITHDRAWAL:
    - The resource is present in the network and is capable of providing the service for the client for a predefined period of time.
      - A temporal expression is used to indicate when the resource will be (temporarily) withdrawn.

    - Applied stereotypes:
        - Preliminary

3.1.2.3 OperationalState

Qualified Name:
CoreModel::CoreFoundationModel::StateModel::TypeDefinitions::OperationalState

The operational state is used to indicate whether or not the resource is installed and working.

Applied stereotypes:

- Mature

Contains Enumeration Literals:

- DISABLED:
  - The resource is unable to meet the SLA of the user of the resource.
    If no (explicit) SLA is defined the resource is disabled if it is totally inoperable and unable to provide service to the user.
  - Applied stereotypes:
    - Mature
- ENABLED:
  - The resource is partially or fully operable and available for use.
  - Applied stereotypes:
    - Mature

3.1.3 Relationship between states in the same context

If the assignmentState is PLANNED then the operationalState must be DISABLED and the administrativeState should be LOCKED.

If the assignmentState is POTENTIAL_BUSY, POTENTIAL_AVAILABLE or UNAVAILABLE the administrativeState should be LOCKED.

If the administrativeState is SHUTTING_DOWN the assignmentState should be PENDING_WITHDRAWAL.

In all other circumstances the states are independent.

3.1.4 Relationship between states in the client context and server context

The tables below list the states in the server context (supporting abstraction) that influence the states in the client context (dependent abstraction) in the same controller.

Administrative state

Supporting abstraction administrativeState	Dependent abstraction administrativeState	Notes
LOCKED	LOCKED	Service is blocked by the supporting abstraction
UNLOCKED	UNLOCKED	
	LOCKED	*Local modification*: Service is not blocked in the supporting abstraction

Operational state

Supporting abstraction operationalState	Dependent abstraction operationalState
ENABLED	ENABLED
DISABLED	DISABLED

Assignment state

Supporting abstraction assignmentState	Permitted dependent abstraction assignmentState	Notes
PLANNED	PLANNED	
INSTALLED	PLANNED	*Local modification:* The administrator may choose to delay showing the change to INSTALLED
	INSTALLED	
	POTENTIAL_AVAILABLE;	*Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource
	POTENTIAL_BUSY	*Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource
	PENDING_WITHDRAWAL_FREE	*Local modification: Controlled by client using the resource*: Resource may be removed from this client. No impact on the supporting abstraction
	SCHEDULED_WITHDRAWAL	*Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource
	SCHEDULED_CAPACITY_CHANGE	*Controlled by the entity managing the supporting abstraction*: Only used if more than one client has a view of the same resource
POTENTIAL_AVAILABLE	POTENTIAL_AVAILABLE	
	PENDING_WITHDRAWAL_FREE	*Local modification*: *Controlled by client using the resource*: Resource may be removed from this client. No impact on the supporting entity

Supporting abstraction assignmentState	Permitted dependent abstraction assignmentState	Notes
POTENTIAL_BUSY	POTENTIAL_BUSY;	
	PENDING_WITHDRAWAL_FREE	*Local modification*: *Controlled by client using the resource*: Resource may be removed from this client.
PENDING_WITHDRAWAL	PENDING_WITHDRAWAL	
	PENDING_WITHDRAWAL_FREE	*Local modification*: *Controlled by client using the resource*: The client is no longer using the resource
UNAVAILABLE	UNAVAILABLE	

No other states in the dependent abstraction (in the client context) have a dependency on the state of the supporting abstraction (in the server context).

The states in the client abstraction (in the server context of the "n+1" controller – as shown in Figure 3-2) should track the states of the dependent abstraction (in the client context of the "n" controller – as shown in Figure 3-2) except for PENDING_WITHDRAWAL_FREE which is controlled by the client using the resource.

The AdministrativeState in the server context is not visible in the client context. The client context may maintain an independent AdministrativeState.

The provider controls the assignmentState of the dependent abstraction that is visible to the client context as described in the table above.

3.1.5 State transition diagrams

These state transition diagrams are preliminary sketches that may be refined in following releases.

3.1.5.1 Administrative State

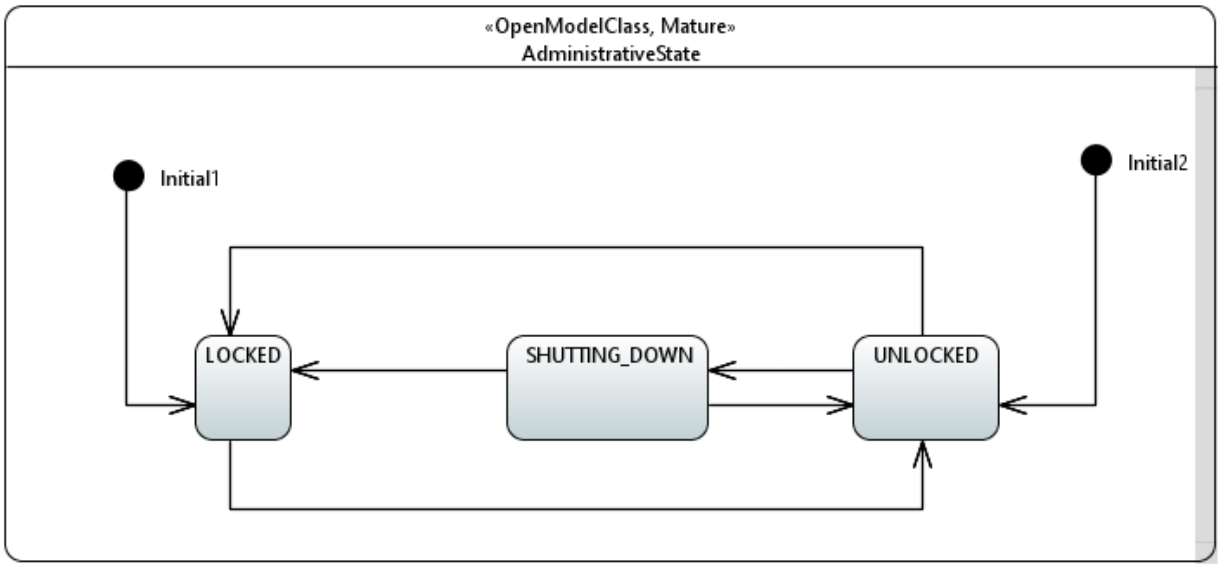


Figure 3-3 Administrative State

The shutting down state is used in a supporting abstraction when a dependent abstraction is in the assignment state of Installed (i.e., a client may be using the resource). The ShuttingDown state is not visible to the dependent abstraction. The assignmentState of the supporting abstraction should be changed to PENDING_WITHDRAWAL when the AdministrativeState transitions to SHUTTING_DOWN. After the assignmentState of the dependent abstraction has transitioned to PENDING_WITHDRAWAL_FREE the AdministrativeState of the supporting abstraction can transition to LOCKED and the assignmentState of the dependent abstraction should be changed to UNAVAILABLE or the abstraction should be deleted.

3.1.5.2 Operational State

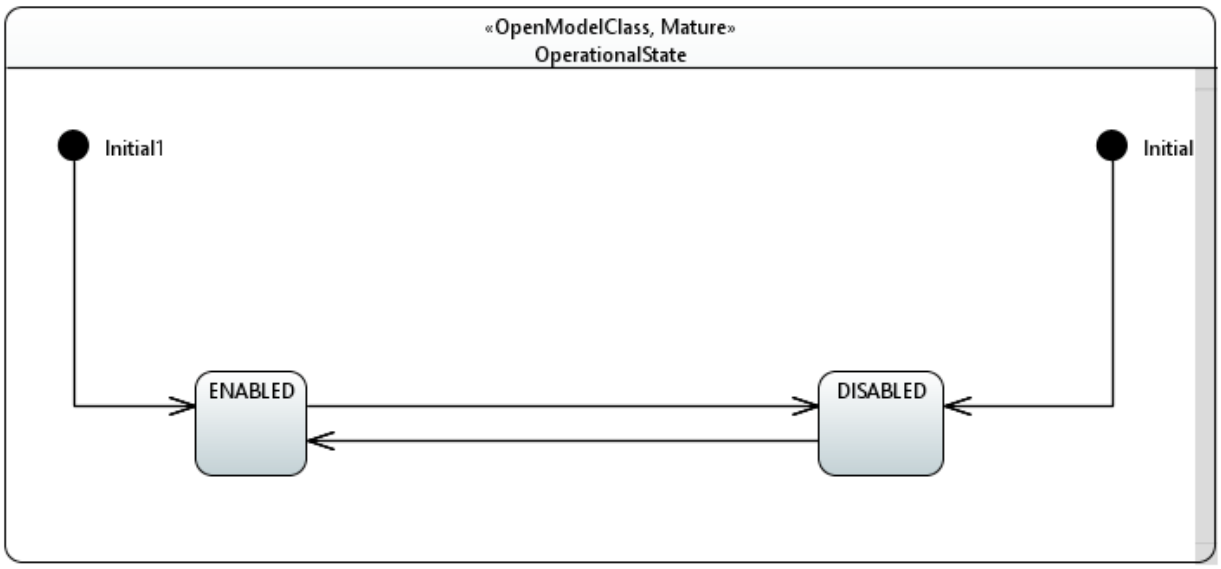


Figure 3-4 Operational State

The operationalState is controlled by the supporting hardware or software and is read only. The operationalState of a dependent abstraction in the INSTALLED state must match the operationalState of the supporting abstraction.

3.1.5.3 Assignment State

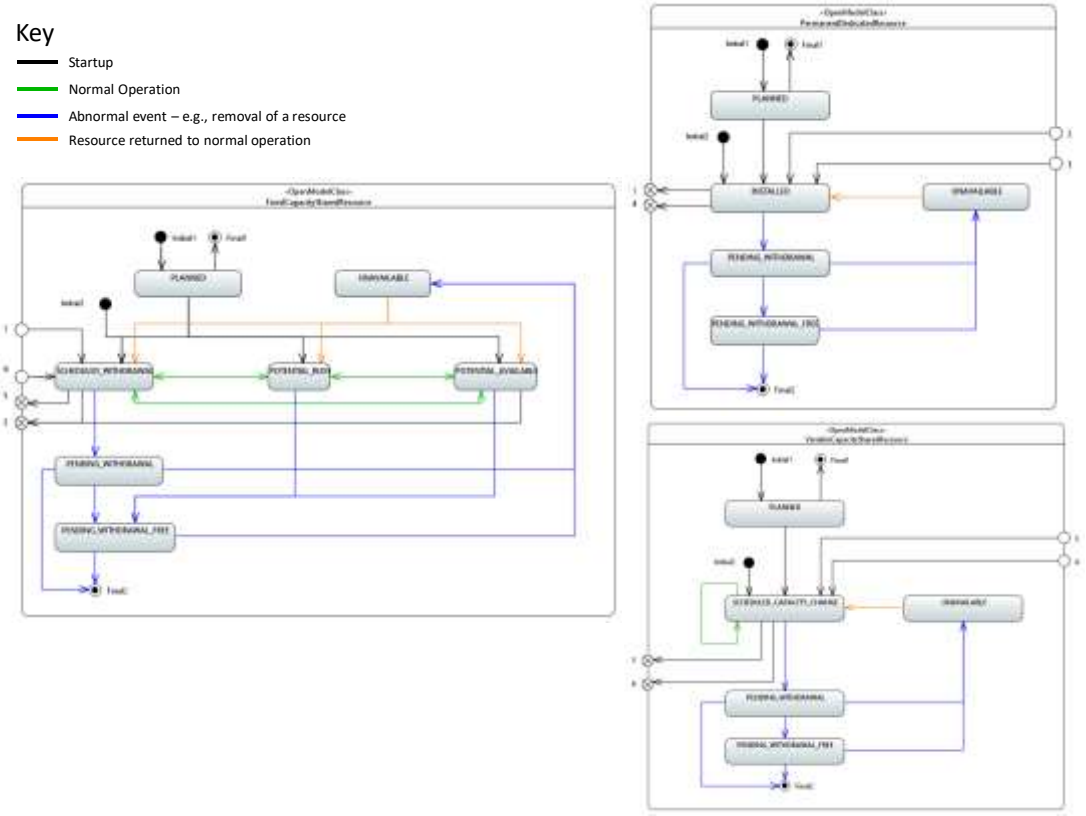


Figure 3-5 Assignment State

The allocation of a resource may be migrated between the use cases as described below.

From PermananetDedicatedResource to FixedCapacitySharedResource (1)

- From INSTALLED to SCHEDULED_WITHDRAWAL

From FixedCapacitySharedResource to PermananetDedicatedResource (2)

- From SCHEDULED_WITHDRAWAL or POTENTIAL_AVAILABLE to INSTALLED

From VariableCapicitySharedResource to PermananetDedicatedResource (3)

- From SCHEDULED_CAPACITY_CHANGE to INSTALLED

From PermananetDedicatedResource to VariableCapicitySharedResource (4)

- From INSTALLED to SCHEDULED_CAPACITY_CHANGE

FixedCapacitySharedResource to VariableCapicitySharedResource (5)

- From SCHEDULED_WITHDRAWAL to SCHEDULED_CAPACITY_CHANGE

From VariableCapicitySharedResource to FixedCapacitySharedResource (6)

- From SCHEDULED_CAPACITY_CHANGE to SCHEDULED_WITHDRAWAL

If a resource is shared by two or more clients (dependent abstractions) when the supporting abstraction changes to INSTALLED, the dependent abstraction transitions to POTENTIAL_AVAILABLE, POTENTIAL_BUSY, SCHEDULED_WITHDRAWAL or SCHEDULED_CAPACITY_CHANGE. A resource may be POTENTIAL_AVAILABLE in all of the dependent abstractions or PENDING_WITHDRAWAL in one dependent abstraction and POTENTIAL_BUSY in all other dependent abstractions.

A resource in a supporting abstraction is moved to PENDING_WITHDRAWAL when the provider of that resource intends to withdraw the resource from the client. The provider should indicate the time when the resource will be withdrawn and indicate if the withdrawal will be permanent or temporary (e.g., to allow network maintenance).

A resource is moved to PENDING_WITHDRAWAL_FREE when the user of that resource is no longer using the resource. This causes the corresponding resource in the client context (the dependent abstraction) to transition to PENDING_WITHDRAWAL_FREE. The provider may withdraw the resource after the specified time even if the client has not transitioned to PENDING_WITHDRAWAL_FREE.

When the resource in the dependent abstraction moves to PENDING_WITHDRAWAL_FREE the administrativeState of the supporting abstraction may be changed to LOCKED. If the withdrawal is temporary, the dependent abstraction should be changed to UNAVAILABLE. If the withdrawal is permanent the abstraction should be deleted.

Note that an abstraction in a server context my play the role of both the client abstraction (when viewed from the perspective of controller n in Figure 3-2) and the supporting abstraction (when viewed from the perspective of controller n+1 in Figure 3-2). In this case the

POTENTIAL_AVAILABLE and POTENTIAL_BUSY states are used in the supporting abstraction.

3.1.6 Use of states

3.1.6.1 Model context

Figure 3-6 below is an informal sketch of the relationship between the "platform," that supports the functions, the logical resource model, that provides a view of the logical resources and the equipment and software models, that provide a view of the implementation platform. Note that this figure shows a highly simplified view of the equipment model and software model, only the major relationships of interest are shown. The client context and server context may be represented by constrain domains. The equipment model is in TR-512.6 and the software model is in TR-512.12.

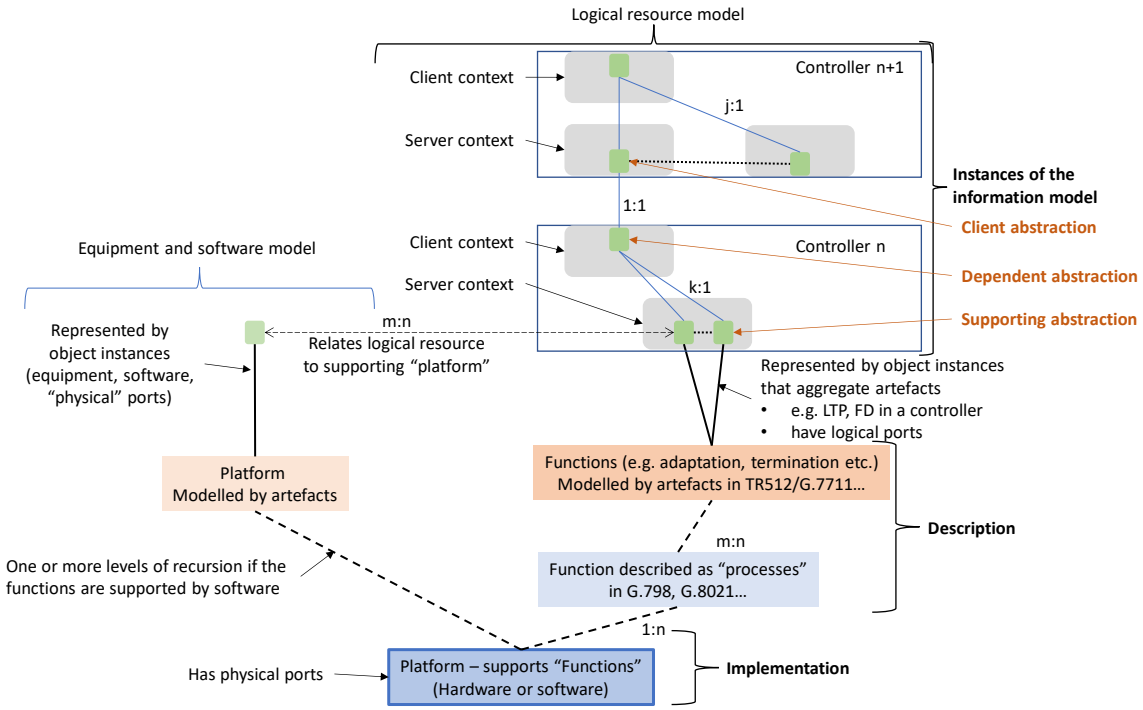


Figure 3-6 Relationship between entities and abstractions

Figure 3-8 illustrates the relationship between the logical resource model and the supporting hardware or software platform, it does not show the full hardware or software model. In the equipment model and software model a field replaceable unit (FRU) or a running software module is represented by an object. These objects support one or more of the artefacts in the logical resource model. In the logical resource model, the lowest level objects are the aggregation of one or more of the artefacts.

Example:

- *An interface card on an Ethernet switch (a single FRU) may support 10 * 1GE interfaces and thus 10 "Ethernet" LTPs.*

- *The PHY of each of the Ethernet interfaces may be on a separate pluggable module (FRU) so each Ethernet LTP is supported by both the interface card (FRU) and the pluggable module (FRU).*

The objects in a server context are referred to as the "supporting abstraction" and the objects in the client context are referred to as the "dependent abstraction". The states of an object in a client context (the dependent abstraction) should be consistent with the state of the corresponding supporting object(s) in the server context (in the same controller). Also, the state of the object in the server context in the n+1 controller (client abstraction) should track the state of the object in the corresponding client context (in the n controller). Note that when the state of one or more of the supporting abstractions changes the state of the dependent abstraction and client abstraction will not be consistent until that change has been processed. In a recursive hierarchy, the client abstraction will play the role of supporting abstraction in the next level of the hierarchy.

Within a controller, the objects "directly" representing the implementation (e.g., in the server context in Controller n in Figure 3-6) should support the Administrative State. When the administrativeState is set to LOCKED, the implementation prevents the resource from being used. Abstraction in any other context (e.g., in the client context in controller n or server context in controller n+1) support of the administrative state is optional. Setting the administrativeState to LOCKED does not have any influence on the resource and it will continue to function normally.

The resources in a server context (i.e., the supporting abstractions) may be aggregated into a single object in a client context that is "mirrored" in the server context of the client controller by the client abstraction.

3.1.6.1.1 Alarms, fault isolation and control

The "lowest level" objects in the resource model are "directly connected" to the implementation and have access to the forwarding namespace. They can configure the resources that implement the logical functions and receive alarm/status information directly from the implementation. Fault location, to the level required for FRU replacement, can only be performed with information from this lowest level which provides access to the resource and physical name spaces.

When an object is presented in a client context, the alarm/status information is abstracted and aggregated, the relationship to the implementation is not visible from the client context.

3.1.6.2 Instance relationships

As described above, the supporting entity may be a "platform" or a managed object. The dependent entity is always a managed object. The relationships between the supporting and dependent abstractions may be:

Number of supporting abstractions/resources	Number of dependent abstractions
1	1
1	n
m	1
m	n

The state of a dependent abstraction must be consistent with the state of the supporting abstraction(s). For the m:1 and m:n cases the states are considered in the order of the precedence described in section 3.2.6.2 below.

The state of the client abstraction (in the server context of controller n+1) should match the state of the dependent abstraction (in the client context of controller n).

3.1.6.2.1 Supporting:Dependent 1:1 Case and 1:n Case

The state of the dependent objects is set based on the state of the supporting abstraction and in some cases may be modified by the local controller as described in 3.1.4. For the 1:n case local modifications are made independently on each of the dependent abstractions.

3.1.6.2.2 Supporting:Dependent m:1 Case

The state of the dependent object is set based on the state of the *composite* supporting instance as described below. The precedence of the states in the tables below are used to determine the state of the *composite* supporting abstraction.

Administrative state

administrativeState	**Precedence**
LOCKED	Highest
SHUTTING_DOWN	
UNLOCKED	Lowest

Operational state

operationalState	**Precedence**
DISABLED	Highest
ENABLED	Lowest

Assignment state

assignmentState	Precedence
PLANNED	Highest
UNAVAILABLE	
PENDING_WITHDRAWAL PENDING_WITHDRAWAL_FREE	
POTENTIAL_BUSY	
POTENTIAL_AVAILABLE	
INSTALLED SCHEDULED_WITHDRAWAL SCHEDULED_CAPACITY_CHANGE	Lowest

The m:1 case has two sub-cases:

3.1.6.2.2.1 Simple

This arrangement of supporting abstraction and composite abstraction is illustrated in Figure 3-7 below.

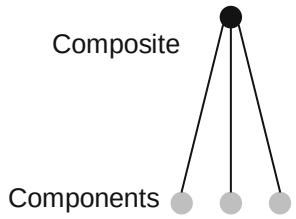


Figure 3-7 Component composite relationship

The state of the composite abstraction is determined by making a list of the states of the component supporting abstractions and arranging that list of states from the lowest to highest precedence. The resource policy defines the number of component supporting abstractions that must be in a given state (or in a lower precedence state) for the composite abstraction to be in that state.

3.1.6.2.2.1.1 Example of the application of resource policy

Using the assignment state enumerations as an example, then in the case of six component supporting abstractions with the following states:

Abstraction	A	B	C	D	E	F
State	PLANNED	INSTALLED	POTENTIAL_AVAILABLE	INSTALLED	POTENTIAL_BUSY	POTENTIAL_AVAILABLE

Rearranging into an ordered list:

	Abstraction	State
1	B	INSTALLED
2	D	INSTALLED
3	C	POTENTIAL_AVAILABLE
4	F	POTENTIAL_AVAILABLE
5	E	POTENTIAL_BUSY
6	A	PLANNED

The state of the composite abstraction is determined by applying the resource policy for the number of component supporting abstractions that must be in a given state (or in a lower precedence state) for the composite abstraction to be in that state. If the policy only requires 1 the composite state will be INSTALLED: If the policy requires 3 the composite state will be POTENTIAL_AVAILABLE; If the policy requires 6 the composite state will be PLANNED.

3.1.6.2.2.2 Compound

This may be modelled by concatenating the appropriate set of simple cases described above. The state of the intermediate composite is evaluated for each (simple) group and is used to define the state of the intermediate component for the next stage of evaluation. That is, each of sets of "lower" components are evaluated, using the rules described above, to determine the intermediate composite states. These intermediate composite states are then used as the intermediate component state when evaluating the state of the next level of composite abstractions. Figure 3-8 below shows an example where three levels of evaluation used to map from the component states into the composite state. The sets of component abstractions that are grouped to form an intermediate composite abstraction is determined by the topological relationship between those abstractions. The compound case could, for example, be used to derive the state of a forwarding domain that is supported by a set of forwarding domains and links (each with its own intermediate composite state).

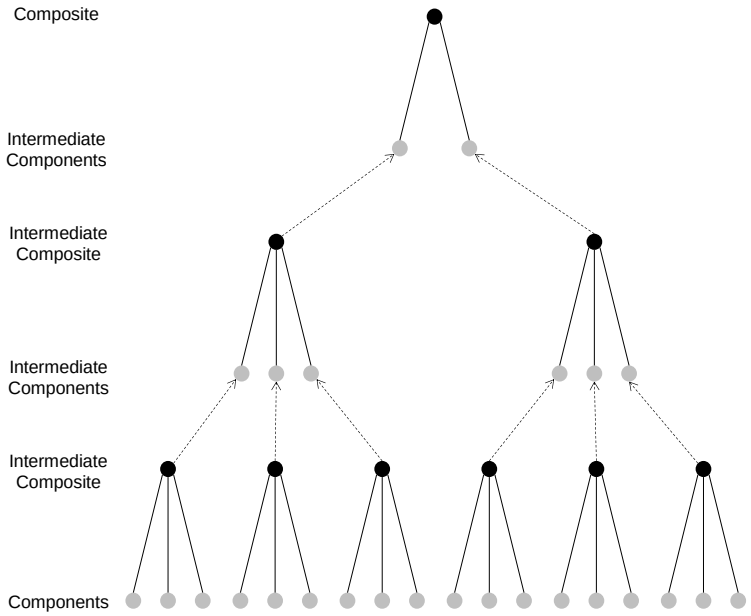


Figure 3-8 Compound component composite relationships

Some (or all) of the intermediate composite abstractions may be exposed to a client. Also, alternate intermediate aggregates of the same component resources may be exposed to a client as illustrated in Figure 3-9 below.

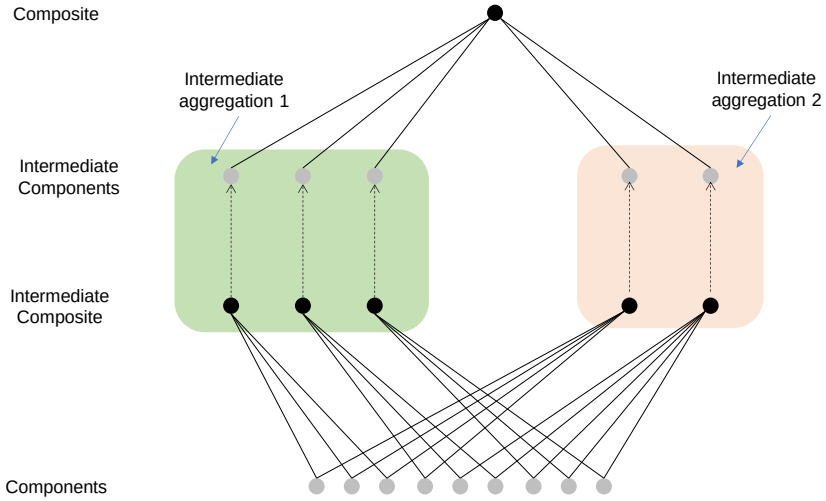


Figure 3-9 Alternate intermediate aggregates

In this case it is important that the state of the composite derived via intermediate aggregation 1 is the same as the state derived via intermediate aggregation 2.

3.1.6.2.3 Supporting:Dependent m:n case

Note: This does not represent a m:n protection scheme.

First the state of the m component abstractions are mapped into a single composite state as described above in 3.1.6.2.2. Then this composite state is used to derive the state of the dependent abstraction as described described above in 3.1.6.2.1. This would be used for example when a forwarding domain that is supported by forwarding domains and links is shared between n clients.

3.1.6.3 Protected entities

The state of an abstraction that is representing a protected resource is determined by the C&SC that is managing/representing the protection scheme. Note that a client (controller) may have a view of both the protected resource and the (unprotected) resources that support it.

3.1.6.4 Split entities

The view (abstractions) presented to a client controller cannot provide a more detailed view than that offered by the lowest level instances that have been created. When a client (controller) view of an entity is "split" as shown in the example in Figure 3-10 below where the original client FD is split into two FDs (X and Y). The links 2, 3 and LTPs 4, 5, 6, 7 are now exposed to the client. The client must retain the context of the original (parent) FD to understand the link between Client FD X and Client FD Y.

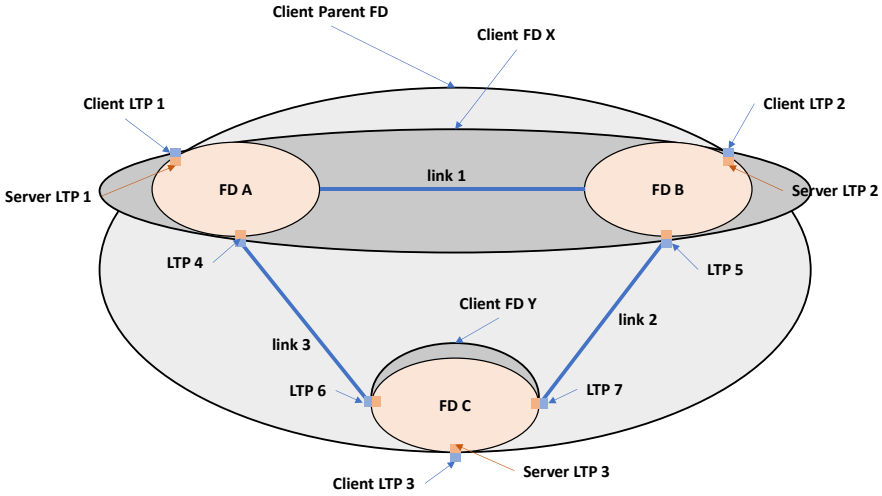


Figure 3-10 Split entity example

The states of the dependent resources provided by the server (controller) must reflect the states of the supporting resources as described above.

The new client FDs should be provided with new identifiers so that the identifier for the parent FD can be retained.

3.1.6.5 Merged entities

The state of the merged entity must be consistent with the state of the supporting resources as described above.

Considering the example above where the server controller is required to "merge" client FD X and client FD Y into a single FD. We have two cases:

- The parent FD is already visible (with an identifier) in the client context, in this case no further action is required.
- The parent FD is not visible in the client context. In this case a new ID for the parent FD should be created. This option must be used if the client has the option of viewing either level of abstraction (i.e., FD X, Y and Parent FD). The ID of either FD X or FD Y could be used for the parent FD. However, this is not recommended since it precludes the possibility of leaving FD X and FD Y visible to the client.

Appendix 1
Examples of the use of temporal expressions for the assignmentState

Example temporal expression for a fixed capacity resource shared by client A and B

	Period 1	**Period 2**	**Period 3**
Server view	Assigned to A	Assigned to B	Free
Client A view	SCHEDULED_WITHDRAWAL	POTENTIAL_BUSY	POTENTIAL_AVAILABLE
Client B view	POTENTIAL_BUSY	SCHEDULED_WITHDRAWAL	POTENTIAL_BUSY*

***** The server may choose not to expose POTENTIAL_AVAILABLE capacity to client B

Example temporal expression for a variable capacity shared resource shared by clients A and B

Packet variable capacity shared resource allocation:

Total capacity = 100:
Σ allocated CIR ≤ 100:
Σ allocated PIR ≤ 200: PIR allocated to one client ≤ 100
Note: in this example the PIR is over-subscribed. Depending on the traffic loading a client may not be able to use the full allocated PIR.

	Period 1	**Period 2**	**Period 3**	**Period 4**
Capacity allocation	Table 1	Table 2	Table 3	Table 4

Table 1		
	CIR	**PIR**
Client A	50	80
Client B	20	30
Unallocated	30	90

Table 2		
	CIR	**PIR**
Client A	0	0
Client B	90	100
Unallocated	10	100

Table 3		
	CIR	**PIR**
Client A	0	10
Client B	20	30
Unallocated	80	160

Table 4		
	CIR	**PIR**
Client A	10	20
Client B	90	100
Unallocated	0	80

TDM variable capacity shared resource

Total capacity = 100:
Σ allocated capacity ≤ 100

	Period 1	**Period 2**	**Period 3**	**Period 4**
Capacity allocation	Table 1	Table 2	Table 3	Table 4

Table 1		
	Allocated	Potential available[1]
Client A	50	30
Client B	20	30
Unallocated	30	-

Table 2		
	Allocated	Potential available [1]
Client A	0	0
Client B	90	10

[1] The server may decide how much of the POTENTIAL_AVAILABLE capacity is exposed to each of the clients

Unallocated	10	-

Table 3		
	Allocated	Potential available[1]
Client A	0	30
Client B	20	50
Unallocated	80	-

Table 4		
	Allocated	Potential available[1]
Client A	10	0
Client B	90	0
Unallocated	0	-

End of document