A semantic model for complex computer networks: the network description language
van der Ham, J.J.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 3

The Network Description Language

This chapter is based on *Using RDF to Describe Networks* by J.J. van der Ham, F. Dijkstra, F. Travostino, H.M.A. Andree and C.T.A.M. de Laat [1] and *Semantics for Hybrid Networks Using the Network Description Language* by J.J. van der Ham, P. Grosso and C.T.A.M. de Laat [2].

3.1 Introduction

In the previous chapter we have examined existing models for describing computer networks. We found that there currently is no suitable model for distribution and pathfinding in a multi-domain scenario. While current information models often use Extensible Markup Language (XML) as portable interchange syntax, we have argued that Resource Description Framework (RDF) is more appropriate to the task.

In this chapter we introduce our information model for topology and network state the Network Description Language. It builds upon RDF and its linking capabilities to produce a distributed view of the global inter-domain network. NDL provides a way to implement the idea of the Topology Knowledge Base as proposed by Travostino in 2005 [61]. To the best of our knowledge NDL is the first ontology in RDF to describe computer networks.
It is worth noting that the proposed network description language is only a method to describe topology information. It does not eliminate the need for a control plane for signalling and provisioning.

In section 3.2 we take a closer look at the problem area of describing networks and in section 3.3 we show the initial version of NDL. Then in section 3.4 we improve the model with simple layering information. Finally in section 3.5 we examine the problem of layer descriptions in more detail, and we further improve the model with a technology independent solution for describing multilayer topologies.

3.2 Terminology for Computer Networks

Terminology plays a very important part in scientific discussions and research. It must identify the important concepts and objects in the problem space. The terminology to describe computer networks and their operations has been the subject of long debates. In 1978 Shoch[62] attempted to define a terminology in order to help discussions on routing in computer networks:

“The “name” of a resource indicates *what* we seek, an “address” indicates *where* it is, and a “route” tells us *how to get there*.”

Unfortunately his paper was little noticed, and it took until 1982 to get more recognition when Saltzer[63] observed that the discussion on computer networking was still confusing because of lack of strict definitions. He took the terminology of Shoch and provided more strict definitions of these terms, and also introduced the term network attachment point, quoting from [63]:

Service and Users These are the functions that one uses, and the clients that use them. Examples of services are one that tells the time of day, one that performs accounting, or one that forwards packets. An example of a client is a particular desktop computer.

Nodes These are computers that can run services or user programs. Some nodes are clients of the network, while others help implement the network by running forwarding services. (We will not need to distinguish between these two kinds of nodes.)

Network attachment points These are the ports of a network, the places where a node is attached. In many discussions about
data communication networks, the term “address” is an identifier of a network attachment point.

**Paths** These run between network attachment points, traversing forwarding nodes and communication links.

Unfortunately, the confusion in discussions on computer networks continued to exist as observed in 1999 by Chiappa[64]. He has examined the use of the term ‘address’, and identifies several different meanings and uses of the term. Chiappa concludes that the overloading of the term ‘address’ has lead to confusion and introduces a more strict definition:

“The name of a network connection entity to which the system of routers will deliver a packet.”

He also introduces the abstract concept of an ‘endpoint’:

“An ‘endpoint’ is thus defined as one participant of an end-end communication; i.e. the fundamental agent of end-end communication. It is the entity which is performing a reliable communication on an end-end basis.”

The main point Chiappa makes is that the address and name for an endpoint should not be the same. When they are different, it becomes possible to have mobile connections, mobile applications, and on the whole makes for a cleaner solution:

“Put in more concrete terms, this argument for explicit recognition of endpoints, and naming of them, says that doing so will result in substantial improvements in overall utility, directness, simplicity, robustness, flexibility, etc; these are all properties which are treasured highly in designs that have to have a long life-time.”

The terminology in the above three papers is mostly aimed at describing routing functionality in computer networks. However, the difficulties encountered there show that describing computer networks is not a trivial problem. Even though computer networks are entirely the product of human engineering, its working has become complex and has grown far beyond easy comprehension.
3.3 The Network Description Language

With the discussions and papers as described in the previous section in mind, we have set out to create an ontology for describing topologies of computer networks. Our contribution is to use the outcomes of the papers and discussions not only to write down clear definitions, but to also make this terminology easily available to applications that want to deal with the network.

Our initial goal has been to describe all the necessary elements for doing pathfinding on a single layer in optical computer networks. The most important elements are then devices, interfaces and how these are connected. A path from one device to another goes through interfaces, connections between interfaces, and in the end to another device.

A second use-case is to provide a good overview of resources. For example, our experimentation network is distributed over two sites with a number of connections between them. The network can be seen as a single network, yet to create an accurate description of both our sites, we use location objects.

Our first schema for the Network Description Language is shown in figure 3.1. An ontology in RDF consists of classes and properties, as we discussed in section 2.5.3. Below we describe the classes and properties of NDL in more detail.

![Diagram of Network Description Language](image)

**Figure 3.1:** The classes and properties of the Network Description Language (version 1)

NDL version 1 has three classes, shown at the top of the figure, which define the kind of resources.

**Location** This class describes a place where resources are located. Often requests for lightpaths are from location to location to connect two computing clusters, or between other sets of resources that are specific to that
location. A Location class is also helpful when drawing network maps, see also section 4.2.

**Device** The physical nodes in the network, this can be any kind of device such as a computing node, a switch, or a router.

**Interface** The interfaces with which devices are connected to a network.

The figure also shows six properties at the bottom, to define the relations between instances of the NDL classes, other classes, or static values.

**locatedAt** A relation between resources and their location,

**hasInterface** A relation between devices and interfaces,

**connectedTo** A relation between two interfaces, describing that they are directly externally connected,

**description** A relation that can be used to include a (human-readable) description of a resource,

**name** To define the name of a resource,

**switchedTo** A relation between two interfaces describing that they are internally connected, for example in optical cross-connects.

The choice for this limited set of classes and properties has been governed by the desire to keep things simple, yet powerful enough to provide accurate descriptions. The language aims to describe the network topology just above the physical layer, so we ignore static physical elements such as filters, or amplifiers. Yet the descriptions provide enough information for our two use-cases, pathfinding and network visualization.

One explicit simplification of the topology has been to describe every element of the network as a generic device. We have also considered differentiating devices by the layer that they operate on, however the actual functionality of networking devices can be very complex. For example, consider current ethernet router/switches. They are capable of both switching and routing, which makes it very hard to explicitly describe their functionality. In the end we concluded that it is important to accurately describe the physical topology. Through the extensibility of RDF, the capability descriptions can always be added later, while retaining full backward compatibility.
A network topology described using these classes and properties contains all elements for pathfinding through the network, if we assume a single technology layer. Even though in optical networks different technologies are used, descriptions in this schema already provide enough information to create network maps, such as the map of the GLIF network shown previously in figure 1.2. These maps give a global overview of the network, and engineers can then get a sense of which domains to contact for connection requests.

An example of a network description is shown in listing 3.1, which describes the network shown in figure 3.2.

```xml
<rdf:RDF

<ndl:Lighthouse #>
<ndl:Lighthouse:name>Rembrandt3</ndl:Lighthouse:name>
<ndl:Lighthouse:location>Rembrandt3</ndl:Lighthouse:location>
<ndl:Lighthouse:interface>Rembrandt3:eth0</ndl:Lighthouse:interface>

<ndl:Lighthouse:interface>Rembrandt3:eth0</ndl:Lighthouse:interface>
<ndl:Lighthouse:connectTo>Glimmerglass:port3</ndl:Lighthouse:connectTo>

<ndl:Lighthouse #>
<ndl:Lighthouse:name>Rembrandt5</ndl:Lighthouse:name>
<ndl:Lighthouse:location>Rembrandt5</ndl:Lighthouse:location>
<ndl:Lighthouse:interface>Rembrandt5:eth0</ndl:Lighthouse:interface>

<ndl:Lighthouse:interface>Rembrandt5:eth0</ndl:Lighthouse:interface>
<ndl:Lighthouse:connectTo>Glimmerglass:port5</ndl:Lighthouse:connectTo>

</rdf:RDF>
```

Figure 3.2: A simple network.

The example in listing 3.1 starts with the standard header of an XML file using an UTF-8 encoding. Lines 2 and 3 open the RDF description, and defines namespaces: rdf is the standard RDF namespace, and ndl is the NDL namespace. The URIs for namespaces in RDF are just identifiers, but often the URI is also used as a URL to publish the schema. In our case we also use http://www.science.uva.nl/research/sne/ndl# to publish the NDL schema.

Line 4 opens the definition of the Lighthouse location. The #-prefix states that the device is defined in the local namespace. Line 5 provides the human-readable name for the location, and line 6 closes the location definition. Lines 7 to 11 define the device Rembrandt3. Line 8 provides a human readable name and line 9 states that this device is located in the location Lighthouse. Finally, line 10 defines that Rembrandt3 has an interface, Rembrandt3:eth0. This interface is defined on lines 12 to 15. The connection to another interface is defined using the connectedTo property on line 14, in this case it is defined to be connected to Glimmerglass:port3. The Glimmerglass device is defined similarly on lines 16–31, and the Rembrandt5 device on lines 32–40. The RDF description is then closed on line 41.

The connection between the Rembrandt3 and the Glimmerglass is defined in both directions. This is used to denote a duplex connection and further ensures the consistency of the description.
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:ndl="http://www.science.uva.nl/research/sne/ndl#"
>
    <ndl:Location rdf:about="#Lighthouse">
        <ndl:name>Lighthouse</ndl:name>
    </ndl:Location>

    <ndl:Device rdf:about="#Rembrandt3">
        <ndl:name>Rembrandt3</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Rembrandt3:eth0"/>
    </ndl:Device>

    <ndl:Device rdf:about="#Glimmerglass">
        <ndl:name>Glimmerglass</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Glimmerglass:port3"/>
        <ndl:hasInterface rdf:resource="#Glimmerglass:port5"/>
    </ndl:Device>

    <ndl:Interface rdf:about="#Glimmerglass:port3">
        <ndl:name>port3</ndl:name>
        <ndl:connectedTo rdf:resource="#Rembrandt3:eth0"/>
        <ndl:switchedTo rdf:resource="#Glimmerglass:port5"/>
    </ndl:Interface>

    <ndl:Interface rdf:about="#Glimmerglass:port5">
        <ndl:name>port5</ndl:name>
        <ndl:connectedTo rdf:resource="#Rembrandt3:eth0"/>
        <ndl:switchedTo rdf:resource="#Glimmerglass:port3"/>
    </ndl:Interface>

    <ndl:Device rdf:about="#Rembrandt5">
        <ndl:name>Rembrandt5</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Rembrandt5:eth0"/>
    </ndl:Device>

    <ndl:Device rdf:about="#Rembrandt5">
        <ndl:name>Rembrandt5</ndl:name>
        <ndl:locatedAt rdf:resource="#Lighthouse"/>
        <ndl:hasInterface rdf:resource="#Rembrandt5:eth0"/>
    </ndl:Device>

    <ndl:Interface rdf:about="#Rembrandt5:eth0">
        <ndl:name>eth0</ndl:name>
        <ndl:connectedTo rdf:resource="#Glimmerglass:port5"/>
    </ndl:Interface>

</rdf:RDF>

Listing 3.1: The NDL description of the network in figure 3.2.
Besides a topology description, the file also describes the current configuration of the Glimmerglass device. The `switchedTo` statement in line 25 states that the `Glimmerglass:port3` has an internal connection to `Glimmerglass:port5`. Just like the `connectedTo` property, the `switchedTo` property must be defined in both directions. The inverse `switchedTo` property from `Glimmerglass:port5` to `Glimmerglass:port3` is given on line 30. With the `connectedTo` and `switchedTo` statements as given above, we have defined a path from the device `Rembrandt3` to `Rembrandt5`.

### 3.4 Extending the Network Description Language

The first version of NDL is very basic and only allows to describe the physical topology of the network. While this is useful in itself, we also want to describe more properties of the network, so that we can do more accurate pathfinding in more complex networks. For that reason we have extended the first version of NDL with extra classes and properties [3].

![Figure 3.3: The classes and properties of the Network Description Language (version 2)](image)

The new schema is shown in figure 3.3. We introduce one new class: `Link`, this class is used to describe connections through a network that operators do not know the exact details of. This is useful to describe trans-oceanic connections; the actual connection is often a leased line provided by a carrier who does not provide a description. There are many trans-oceanic links, sometimes even multiple between the same locations, so it is important to be able to distinguish between them.
On the other hand a Link object can also be used to provide a more detailed description of an internal connection, because it allows operators to break up the connection in more detailed parts. An example of this is shown later.

We also add two new properties to allow for the description of layer information to provide support for multi-layer pathfinding: encodingType and transportType. These two properties can be associated with objects of the Link or Interface class to describe layering information similar to GMPLS[65].

The choice for using the layer descriptions of GMPLS has been governed by the fact that GMPLS provides a very detailed model for describing the encodings and capabilities of devices in multi-layer networks. GMPLS is also the de facto standard for intra-domain circuit provisioning.

<table>
<thead>
<tr>
<th>Type</th>
<th>GMPLS Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet</td>
<td>1</td>
</tr>
<tr>
<td>Ethernet</td>
<td>2</td>
</tr>
<tr>
<td>ANSI/ETSI PDH</td>
<td>3</td>
</tr>
<tr>
<td>SONET/SDH</td>
<td>5</td>
</tr>
<tr>
<td>Digital Wrapper</td>
<td>7</td>
</tr>
<tr>
<td>Lambda (photonic)</td>
<td>8</td>
</tr>
<tr>
<td>Fiber</td>
<td>9</td>
</tr>
<tr>
<td>FiberChannel</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3.1: Description of values for the encodingType property

Table 3.1 shows the possible values for the encodingType property as defined by the Internet Engineering Task Force (IETF). NDL uses the same values as defined for GMPLS. The lambda encoding type refers to interfaces that use whole wavelengths, such as wavelength selective switched. The fiber encoding refers to interfaces that encode the whole fiber, such as in an optical cross connect.

The values of the transportType property define the technology used to map the data on to the encoding. For example, value 18 denotes ‘Byte synchronous mapping of DS1/T1’, which can be used on encodingType 5 (SDH). The values of the transportType property are given in table 3.2.

Furthermore we extend NDL with another property for defining the capacity of a Link or Interface using the capacity property. There we also follow the definition of GMPLS and use bytes per second as the unit of the capacity, where the values are given using the IEEE floating point format[66].

We follow the definition of capacity as used in GMPLS[65]. That is, the
<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unknown</td>
<td>All</td>
</tr>
<tr>
<td>5</td>
<td>Asynchronous mapping of E4</td>
<td>SDH</td>
</tr>
<tr>
<td>6</td>
<td>Asynchronous mapping of DS3/T3</td>
<td>SDH</td>
</tr>
<tr>
<td>7</td>
<td>Asynchronous mapping of E3</td>
<td>SDH</td>
</tr>
<tr>
<td>8</td>
<td>Bit synchronous mapping of E3</td>
<td>SDH</td>
</tr>
<tr>
<td>9</td>
<td>Byte synchronous mapping of E3</td>
<td>SDH</td>
</tr>
<tr>
<td>10</td>
<td>Asynchronous mapping of DS2/T2</td>
<td>SDH</td>
</tr>
<tr>
<td>11</td>
<td>Bit synchronous mapping of DS2/T2</td>
<td>SDH</td>
</tr>
<tr>
<td>13</td>
<td>Asynchronous mapping of E1</td>
<td>SDH</td>
</tr>
<tr>
<td>14</td>
<td>Byte synchronous mapping of E1</td>
<td>SDH</td>
</tr>
<tr>
<td>15</td>
<td>Byte synchronous mapping of 31 * DS0</td>
<td>SDH</td>
</tr>
<tr>
<td>16</td>
<td>Asynchronous mapping of DS1/T1</td>
<td>SDH</td>
</tr>
<tr>
<td>17</td>
<td>Bit synchronous mapping of DS1/T1</td>
<td>SDH</td>
</tr>
<tr>
<td>18</td>
<td>Byte synchronous mapping of DS1/T1</td>
<td>SDH</td>
</tr>
<tr>
<td>19</td>
<td>VC-11 in VC-12</td>
<td>SDH</td>
</tr>
<tr>
<td>22</td>
<td>DS1 SF Asynchronous</td>
<td>SONET</td>
</tr>
<tr>
<td>23</td>
<td>DS1 ESF Asynchronous</td>
<td>SONET</td>
</tr>
<tr>
<td>24</td>
<td>DS3 M23 Asynchronous</td>
<td>SONET</td>
</tr>
<tr>
<td>25</td>
<td>DS3 C-Bit Parity Asynchronous</td>
<td>SONET</td>
</tr>
<tr>
<td>26</td>
<td>VT/LOVC</td>
<td>SDH</td>
</tr>
<tr>
<td>27</td>
<td>STS SPE/HOVC</td>
<td>SDH</td>
</tr>
<tr>
<td>28</td>
<td>POS - No Scrambling, 16 bit CRC</td>
<td>SDH</td>
</tr>
<tr>
<td>29</td>
<td>POS - No Scrambling, 32 bit CRC</td>
<td>SDH</td>
</tr>
<tr>
<td>30</td>
<td>POS - Scrambling, 16 bit CRC</td>
<td>SDH</td>
</tr>
<tr>
<td>31</td>
<td>POS - Scrambling, 32 bit CRC</td>
<td>SDH</td>
</tr>
<tr>
<td>32</td>
<td>ATM mapping</td>
<td>SDH</td>
</tr>
<tr>
<td>33</td>
<td>Ethernet</td>
<td>SDH, Lambda, Fiber</td>
</tr>
<tr>
<td>34</td>
<td>SONET/SDH</td>
<td>Lambda, Fiber</td>
</tr>
<tr>
<td>35</td>
<td>Reserved (SONET deprecated)</td>
<td>Lambda, Fiber</td>
</tr>
<tr>
<td>36</td>
<td>Digital Wrapper</td>
<td>Lambda, Fiber</td>
</tr>
<tr>
<td>37</td>
<td>Lambda</td>
<td>Fiber</td>
</tr>
<tr>
<td>38</td>
<td>ANSI/ETSI PDH</td>
<td>SDH</td>
</tr>
<tr>
<td>40</td>
<td>Link Access Protocol SDH</td>
<td>SDH</td>
</tr>
<tr>
<td>41</td>
<td>FDDI</td>
<td>SDH, Lambda, Fiber</td>
</tr>
<tr>
<td>42</td>
<td>DQDB (ETSI ETS 300 216)</td>
<td>SDH</td>
</tr>
<tr>
<td>43</td>
<td>FiberChannel-3 (Services)</td>
<td>FiberChannel</td>
</tr>
<tr>
<td>44</td>
<td>HDLC</td>
<td>SDH</td>
</tr>
<tr>
<td>45</td>
<td>Ethernet V2/DIX (only)</td>
<td>SDH, Lambda, Fiber</td>
</tr>
<tr>
<td>46</td>
<td>Ethernet 802.3 (only)</td>
<td>SDH, Lambda, Fiber</td>
</tr>
</tbody>
</table>

**Table 3.2:** The possible values of the `transportType` property (from [65])
capacity includes the header space and is in bytes per second. For example, the capacity of a 10 Gigabit Ethernet Link is given as \(0x4E9502F9\) in IEEE floating point. This translates to \(1.25 \times 10^9\) bytes per second, which is equal to \(10^{10}\) bits per second.

### 3.5 The Multi-Layer Network Description Language

In the previous section we have shown a first attempt at describing multiple layers in NDL. The definitions given there are not yet complete, it is only possible to describe the transport layer, and not the switching layer. The definitions used in GMPLS to describe this are quite complex, and even require splitting the description of a device in certain cases\(^1\). Another disadvantage of using the GMPLS method is that it requires the definition of the technologies in the schema. A new technology means that it cannot be described until there is an updated schema.

While GMPLS is the de facto standard in practice, the ITU-T G.805\(^68\) provides a strong theoretical foundation for describing network technologies and the relations between them. Below we provide a brief introduction to the theoretical foundation of G.805 and show how we have mapped this to NDL. We also extend NDL further with more classes and properties to be able to describe more details of networks. We have split up multi-layer NDL in several schemata, in the following subsections we first explain the new topology schema, then the layer schema, the capability schema and finally the domain schema.

#### 3.5.1 NDL Topology Schema

The multi-layer topology schema is an updated version of the previous NDL schema. We have added several new classes:

**Virtual Device** The resources of physical devices are often split up using virtualization. One physical device can host several virtual devices.

**Static Interface** An interface which is fixed and not configurable.

**Configurable Interface** An interface whose label can be dynamically configured. For example a tuneable laser.

\(^1\)See for example section 4.2.1 in Request For Comments (RFC) 5212\(^67\) which shows how to describe a node that is capable of doing both packet-switching and time-division multiplexing.
Figure 3.4: Classes and predicates in the NDL topology schema.
Interface The Interface class is now used to describe a configured interface, i.e. a particular configuration of a Configurable Interface.

Broadcast Segment A generic case of a Link. A Broadcast Segment is a direct (not concatenated) connection between multiple Interfaces. A Link can only connect two Interfaces, while a Broadcast Segment connects multiple interfaces.

Cross Connect A collection of network elements that can be represented as a subnetwork connection (ITU-T G.805 terminology). A Cross Connect is an internal data transport within a Device, unlike Links which transport data between two Devices.

Path A collection of network elements that can be represented as a tandem connection (ITU-T G.805 terminology) or as a path in a graph (in graph theory). A path is always a connection at a single layer.

We have also introduced three new properties and changed definitions of several others. In the new schema we have introduced the Path concept to describe Paths. Along with this we have also changed the way of describing connections between Interfaces.

A direct connection between two Interfaces (on the same layer) is described either by linkTo statements between two Interfaces, or to an intermediate Link object. The linkTo property is taken to be a uni-directional connection. Similarly, the internal links are described using the switchedTo property, either directly or by using an intermediate Cross Connect object.

The connectedTo property is used to define an external connection between two Interfaces when the details are not all known. Connections can be defined directly or using an intermediate Path object.

The path segments property is used to describe a Path. The Path object is a container object with a sequence of paths and links.

In this new schema we have removed the encodingType and transportType properties. We have deprecated these in favour of the ITU-T G.805 approach described with the Layer schema in the next section.

3.5.2 NDL Layer Schema

The ITU-T G.805 document provides an extensive definition of details of adaptations between different technologies. Clients of the network only require knowledge about a subset of these definitions, so below we provide a simplified
version. For a more extensive introduction to G.805 and its relation to NDL see [4], [13].

At the core of the definition is an adaptation function, this function defines how data is taken from a higher layer, the client layer, and adapted into a lower layer, the server layer. An adaptation function is always bi-directional, so it also defines how data is de-adapted from the server layer to the client layer. Finally, the function defines the adaptation of a specific client layer to a specific server layer, and it is possible to have multiple adaptation functions for the same layer tuple.

![Diagram](image)

**Figure 3.5:** A simplified graphical representation of an adaptation function (left) and a multiplexing function (right)

Figure 3.5 shows the graphical notation. On the left we show a simple adaptation function. On the right is an example of multiplexing: multiple client layers, channels, are adapted together into a single server layer. In order to demultiplex to the original channels labels are required, for example VLAN numbers or wavelengths in a WDM signal. The reverse is also possible, i.e. one client layer into multiple server layers, this is called inverse multiplexing.

Layers are described in NDL using logical interfaces. This means that a single physical interface is described by multiple Interface objects, each on a different layer, depending on the properties of that interface. For example a physical interface in an Ethernet switch will have a logical interface on the physical layer, and on the Ethernet layer, with an adaptation between them. A more extensive example is described below in listing 3.2.

In figure 3.6 we show the NDL layer schema based on the G.805 model. The layer schema does not define actual adaptation functions, but instead provides
Figure 3.6: Classes and predicates in the NDL layer schema.
a common vocabulary to describe technologies, layers and the relation between layers.

Adaptation functions are defined using the class `Adaptation Property`. The definition of that function is given using four properties: the client layer, the server layer, the `client count` and the `server count`.

The client and server layer properties are not explicitly defined as such, instead they are given as the `rdfs:domain` and `rdfs:range` of that specific `AdaptationProperty` instance.

The `client count` represents the maximum number of client layer interfaces. The `server count` represents the number of required server layer interfaces. For one-to-one adaptations, the client count and server count are both one. For multiplexing adaptations, the server count is set to one, and the client count is greater than one, see the example in listing 3.2. For inverse multiplexing adaptations there is a single client layer, transported over multiple server layer connections, for example a 1 Gigabit Ethernet connection that is transported using 21 STS channels. For such an adaptation the client count is 1, and the server count is 21.

A Layer is a specific encoding, or a set of compatible encodings. Associated with a layer is a `Label Set`, the set of labels allowed for that layer. For example the label used for the Ethernet Layer is the VLAN, which must come from the set of integers \{0, 1, 2, ..., 4095\}.

The set of labels that are allowed on an interface are described using the `ingress label set` and `egress label set` properties. The property `label set` is shorthand for setting both ingress and egress to the same value. The actual labels configured on an interface are described using the `ingress label` and `egress label` properties, with a similar `label` shorthand.

Finally we also define `ingress property` and `egress property` for layer specific properties of an Interface. These properties are important to describe, as they may cause incompatibilities between Interfaces on the same layer. For example the MTU size of an Ethernet Interface: normally this is 1500 bytes, but in some cases it is configured to a higher setting.

An example multi-layer description is given in listing 3.2. Lines 1 to 6 start the XML RDF description and define namespaces. Besides the `rdf` and `ndl` namespaces, we also define the RDF Schema (`rdfs`) namespace to use some extra RDF properties, and we include the `layer` and `wdm` schemata. Lines 7 to 12 show an excerpt of that `wdm` schema with the definition of the `WDM adaptation property`. Line 7 defines that it is an `AdaptationProperty` and line 8 defines that it is also a regular RDF property. Lines 9 and 10 define the domain and range of the adaptation property, in this case `FiberNetworkElement` and
Listing 3.2: The NDL description of a WDM adaptation.
LambdaNetworkElement respectively. For this adaptation property we define that the serverCount is 1 (an XML Schema integer) on line 11. We do not define the clientCount, because this is variable for WDM. WDM is a multiplexing adaptation that is always transported over a single fiber (the server layer). The client count is dependent on the specific implementation and configuration of WDM on a device.

A label property is defined on lines 13 to 16. It is a regular RDF property, as defined on line 13, but also a kind of NDL label, as defined on line 14. The range of the label is an XML Schema float number. This label is used in an interface, port3-l1310, that is defined in lines 17–20. Line 17 defines the NDL Interface, and line 18 defines that it is on the Lambda layer. The wavelength of the interface, 1310 nm, is defined on line 19. Another interface, port3, is defined on lines 21–24, line 21 states that it is an NDL Interface, and line 22 defines that the interface is on the Fiber layer. Everything is tied together on line 23, there we define that the port3-l1310 interface is adapted to the port3 using the WDM adaptation.

### 3.5.3 NDL Capability Schema

In the previous section we have defined classes and properties to describe how data is encoded in the network. In this section we define how to describe the capabilities of networking devices, that is how they move data coming in from one interface out to another interface.

![Switch Matrix Diagram](image-url)

**Figure 3.7:** The SwitchMatrix class and its related properties

A Switch Matrix represents the switching capability of a device or domain at a single layer. If a domain or switch can operate on multiple layers it may have multiple switching matrices: one for each layer. A switch matrix can be statically configured to forward data from one logical interface to another logical...
interface. These configurations are represented by a `switchedTo` property in NDL.

The capabilities of a switch matrix are defined using three properties:

**Switching capability** The *switching capability* describes the ability of a device to forward data from one interface to another interface with the *same* label. Two interfaces without a label are considered to have equal labels—both the “empty” label.

**Swapping capability** Some switch matrices are also able to convert between labels. For example, some Ethernet switches can change the VLAN tag. The *swapping capability* represents the ability of a device to forward data from one interface to another interface with a *different* label.

**Cast type** The cast type defines how a switch matrix can switch different interfaces. Most switch matrices can only do *unicast*, i.e., only cross connect two unused interface.

Some switch matrices can do *multicast*: they can make a cross connect from A to B, even if there already is another cross connect with source A.

*Broadcast* switch matrices are entirely different: if two interfaces have the same label, then they must exchange data. An example of this is an Ethernet switch matrix that switches on VLAN labels.

### 3.5.4 Domain Schema

The topology schema allows the description of physical network topologies. The NDL domain schema allows to group these descriptions in networks.

The two classes in the domain schema are:

**NetworkDomain** is a collection of network elements. It behaves very similar to a Device in the topology schema, but describes a domain rather than a physical device.

**AdministrativeDomain** is an organizational entity that is responsible for the operational control of resources (including network resources).

### 3.5.5 Technology Independence

Each network can independently describe their own network in NDL. Similarly, we want our multi-layer network description to be technology independent.
A network description relies on the topology and specific technology schemata. The technology schemata rely on the layer schema.
by providing building blocks to describe technologies. In the multi-layer NDL schemata we have provided classes and properties to describe the details of layers and adaptations. This is a clear de-coupling of topology and technology information.

Figure 3.8 shows our implementation of the de-coupling between topology and technologies. It shows a network topology description at the top-left, which makes use of the topology schema (bottom-left), and a specific technology schema, in this case WDM (top-right). This technology schema is defined using the NDL Layer schema (bottom-right). These descriptions are all tied together using standard RDF definitions.

We have created examples of technology schemata for Ethernet, WDM or TDM and more[69]. The technology schemata are defined as subclasses or instances of classes defined in the layer schema. A path finding algorithm should only have knowledge of the topology schema and layer schema, and learn about a specific network or about specific technologies by reading specific descriptions based on these schemata. This approach allows for a pathfinding application that learns about layers and technologies from the NDL descriptions. This means that the application does not have to know a priori about all the technology details[70].

### 3.5.6 Comparing NDL and GMPLS

In our previous attempt at extending NDL to allow multi-layer descriptions in section 3.4 we have used concepts from GMPLS with the `encodingType` and `transportType` properties and values. However, we soon realised that this approach requires us to statically define the layers and technologies. With the ITU-T G.805 approach described above, we can describe layers and adaptations independently. The multi-layer topology descriptions can then link to the relevant layer and adaptation details. This allows for a more dynamic and future-proof approach.

However GMPLS is currently the de facto standard in the management of optical networks. Our multi-layer NDL should be able to describe anything that is possible to describe using GMPLS. In order to prove this we have attempted to map the information that can be carried in OSPF(-TE) LSAs to NDL. We have concluded that it is possible to translate all the relevant multi-layer topology information from OSPF and OSPF-TE to NDL. The details of these mappings can be found in appendices A and B respectively.
3.6 Conclusion

In this chapter we have introduced the way we apply RDF to describing networks: the Network Description Language. At first NDL only provided a way to describe the physical topology. This allowed us to create linked multi-domain topologies, which can be very useful for network diagrams, and as aid to engineers in manual pathfinding.

In the next step we have extended NDL towards multi-layer network descriptions. Building on the work of GMPLS, we extended NDL with properties to describe encoding and transport types, as described in section 3.4. However, the descriptions as inspired by GMPLS do not allow us to describe arbitrary technologies. When new technologies are introduced, the schema must be updated to include new values for these technologies.

Believing that there must be a more optimal solution, we turned to the ITU-T G.805 standard. ITU-T G.805 provides a way to generically describe layers and adaptations. Building on these ideas, we defined the NDL Layer schema as shown in figure 3.6. Taken together with the Capability schema, and the Domain schema, we now have a good solution for describing multi-layer multi-domain networks with at least the same expressivity as GMPLS.

A complete overview of the NDL schemata as a UML class diagram is shown in figure 3.9.

In section 2.2 we set requirements for a network model, it should be concise, interoperable, distributed, portable, and extensible. In this chapter we have shown the definitions of the NDL schemata, which provide concise definitions for how to use each class and property. By using RDF as the data model for NDL, it is interoperable, following an open standard. It is also distributed, with the powerful seeAlso property, it is possible to link together different, separately maintained network descriptions. RDF also makes it easy to create globally unique identifiers using the automatic prefixing with the # notation, this makes network descriptions easily portable. RDF also allows to easily extend descriptions with properties from other schemata. Finally, RDF can be expressed in XML or other human-readable syntaxes. For these reasons NDL satisfies all of our original requirements.
Figure 3.9: UML representation of the complete NDL schema