Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS

Bc. Elena Myazina

Tools for NDL Elaboration

Department of Software Engineering

Supervisor of the master thesis: doc. Ing. Karel Richta, CSc.

Study program: Computer Science

Specialization: Software Systems

Prague 2014

I would like to thank my supervisor, doc. Ing. Karel Richta, CSc. for his patience and for his valuable expert advices. Moreover, I would like to thank my parents and my friends for supporting me during my studies.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Praque, July 30, 2014 Elena Myazina

Název práce: *Nástroje pro zpracování NDL*

Autor: *Bc. Elena Myazina*

Katedra (ustav): *Katedra softwarového inženýrství*

Vedoucí diplomové práce: doc. Ing. Karel Richta, CSc., katedra softwarového inženýrství, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze

Abstrakt: *Aktuální stav ve výzkumu sítí umožňuje koncovým uživatelům vytvářet vlastní specifická spojení (lightpaths) pro danou aplikaci a optické privátní sítě (OPNs). Taková nastavení vyžadují jasnou komunikaci mezi žádající aplikací a požadovanou sítí. Jazyk NDL (Network Description Language) je určen pro popis takových optických sítí. Je založen na Resource Description Framework (RDF). Tato práce se zabývá analýzou popisu sítě v jazyce NDL a jeho zpracováním. Aktuální stav sítě lze načíst ve formátu NDL a zobrazit jeho grafickou reprezentaci. V tomto stavu lze hledat cesty v reprezentované síti, případně provádět úpravy sítě. Výsledek pak lze opět transformovat do formátu NDL.*

Klíčová slova: *NDL, RDF, optická technologie*

Title: *Tools for NDL Elaboration*

Author: *Bc. Elena Myazina*

Department: *Department of Software Engineering*

Supervisor of the master thesis: doc. Ing. Karel Richta, CSc., Dept. of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague

Abstract: Current state of research of networks enables end users to create custom connections (lightpaths) for a given application and optical private networks (OPNs). Such adjustments require clear communication between the requesting application and the desired network. Language NDL (Network Description Language) is used to describe such optical networks. NDL is based on the Resource Description Framework (RDF). This thesis analyzes the network description language NDL and its treatment. Current status of the network can be loaded in the form of the description in the NDL and displayed as the graphical representation. The representation enables to find out paths in the network, or edit the network. The result can then be transformed back into NDL format.

Keywords: *NDL, RDF, optical technology*

Contents

Chapter 1

Introduction

Computer networks seems to be integral part of current society. Many broadly used public systems are based on the communication networks as a part of distributed processing of information. Newest applications require underlying physical infrastructure of wires, fibers, switches and routers provides network services for applications on the network.
One important problem with such networks is to find appropriate paths through such a network, satisfying some criteria – distance, time constraints, permeability, etc. For this purpose tools are needed for network configuration elaboration. Any such tool has to elaborate a formal description of the network. This formal description serves as a definition of the network structure, and contains all necessary information about network nodes and connections.
Network Description Language (NDL) is designed as a tool for the specification of fiber nets. NDL serves for formal description of network elements and combining such elements into more complex nets.
The aim of this thesis is to design and develop a tool for elaboration of specifications of networks in NDL. The tool has to support creation and verification of such specifications - e.g. queries, if two points in the net are connected, and what are real parameters of this connection.

Thesis organization

The thesis is organized in the following way: chapter 1 is an introduction to the problem, chapter 2 describes the NDL language, the chapter 3 contains the description of the NDL application (here I will often refer to the scientific works of the two authors [35] [36]), and the user guide to this application is at the chapter 4. The chapter 5 discusses the problem of the shortest path in the network, and appropriate algorithms. The selected algorithms is the part of the final solution. The chapter 2 describes the general architecture of my project. The chapter 7 analyzes existing solutions, chapter 8 summarizes the work. The last chapter, Appendix A, lists the contents of the attached CD.

1.1 Computer Networks

"Communication networks are ubiquitous in our society: we use them to make phone calls, end e-mails, and browse the web. In all cases an underlying physical infrastructure of wires, fibres, switches and routers provides network services for applications on the network [32].
Network services can be classified as circuit switched and packet switched network services. The public switched telephone service (PSTN), also known as plain old telephone service (POTS), is a circuit based switching technology" [36].
"The Internet on the other hand is largely based on packet switched technology.
Creating connections in a circuit-switched networks, and especially optical networks, is a complex task. A complete end to end circuit must be created before the connection can be used. Provisioning such a circuit requires knowledge of the network topology, capacities and capabilities. This thesis describes the Network Description Language (NDL), an ontology for describing complex network topologies and technologies. The language defines a clear terminology for describing network topologies that can be linked to other descriptions, to other kinds of resources, but also to other networks, creating a distributed interoperable description of the global topology.
Network operators tend to be protective of detailed topology information, because of scalability, security, or policy reasons. It is also possible to share an aggregated view of the topology, so that only the most important details are published" [35].

1.2 Hybrid Networking

"The idea of providing e-science applications with deterministic point-to-point connections was fostered by a community of research networks, later organised in the Global Lambda Integrated Facility (GLIF)[1]. This community provides a global network to support data-intensive scientific research, and also supports middleware development for optical networking. The ideas in this community led to the concept of hybrid networking, the offering of packet switched (IP) services and circuit switched connections over the same physical network infrastructure.

Since most e-science applications operate in a large scale environment, with collaborators at different universities, the networks required for these applications are nearly always multi-domain networks. De Laat estimated in 2000 that a typical network connection for a physics experiment crosses seven domains [2].
To achieve inter-domain operation, the different networks have to collaborate.

For dedicated network connections, this collaboration is done in the GLIF community [35].

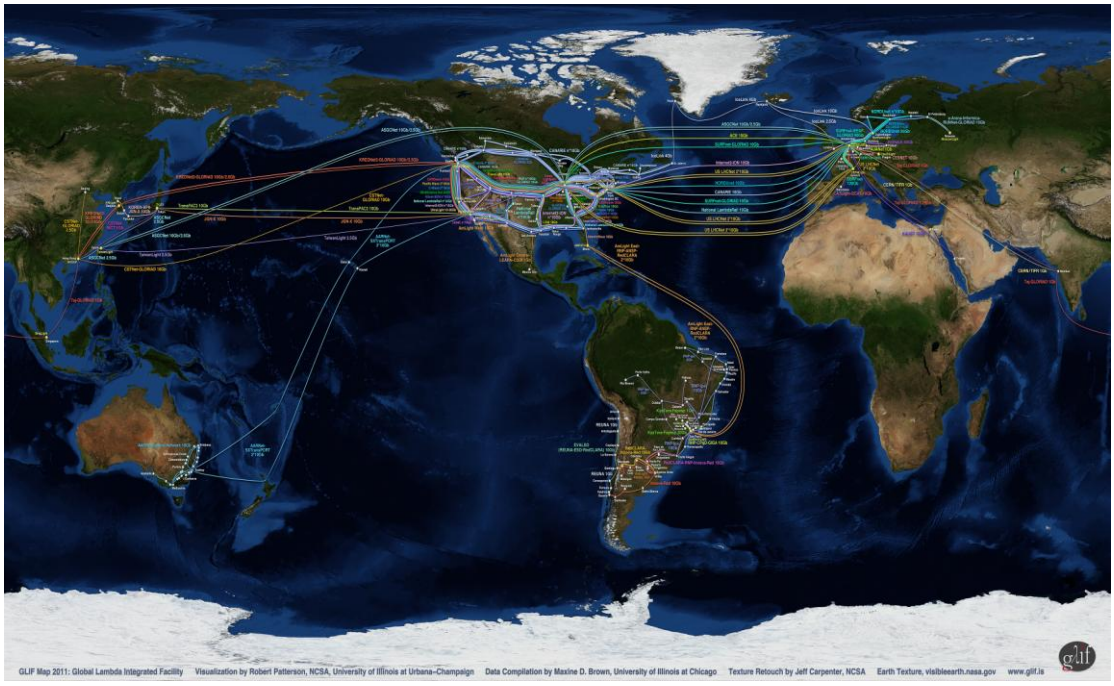


FIGURE 1.1: GLIF WORLD MAP OF MAY 2011

In few years time a number of international network connections have been established to provide the inter-domain connectivity. Figure 1.1 shows a collection of the interconnections provided by partners in the GLIF community as of May 2011" [35].
The GLIF community is working hard at improving the light path provisioning process by exchanging experiences, documenting processes and developing middleware.

1.3 Network Management

In computer networks, network management refers to the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems. Network management is essential to command and control practices and is generally carried out of a network operations center.
Functions that are performed as part of network management accordingly include controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a network, network planning, frequency allocation, predetermined traffic routing to support load balancing, cryptographic key

distribution authorization, configuration management, fault management, security management, performance management, bandwidth management, Route analytics and accounting management.
Data for network management is collected through several mechanisms, including agents installed on infrastructure, synthetic monitoring that simulates transactions, logs of activity, sniffers and real user monitoring. In the past network management mainly consisted of monitoring whether devices were up or down; today performance management has become a crucial part of the IT team's role which brings about a host of challenges—especially for global organizations [34] .
I describe the current architecture for management of computer networks. Figure 1.2 shows a schematic overview of this architecture.

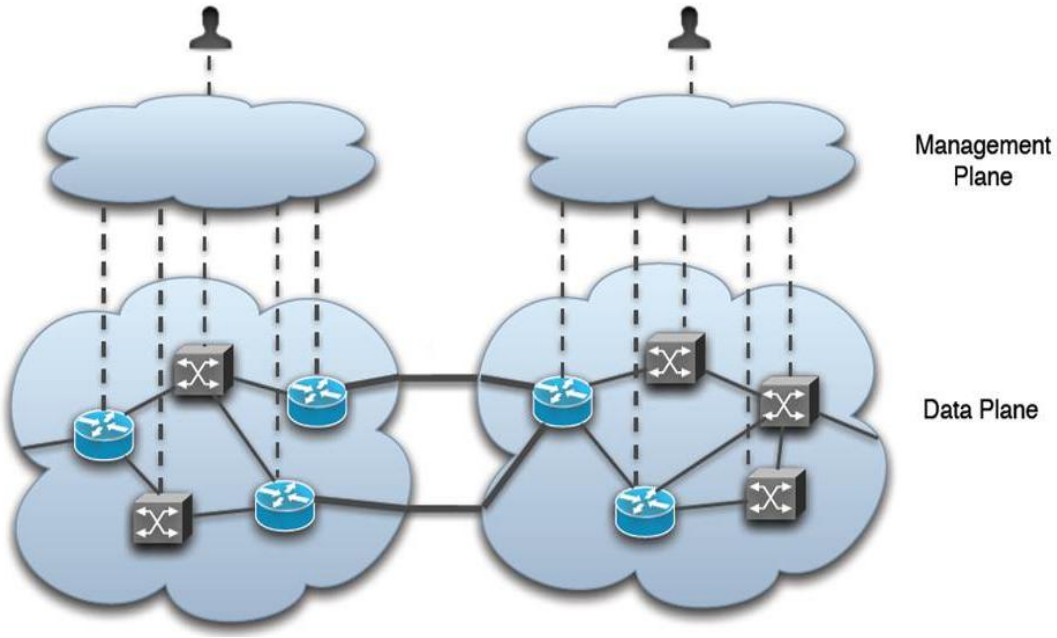


FIGURE 1.2: THE MANAGEMENT PLANE (TOP) AND THE DATA PLANE (BOTTOM) [36].

"At the bottom is the data plane. This is the physical network over which the signals are sent using data communication protocols, electric or optical pulses.
The data plane takes care of moving the data from source to destination. Examples of data plane implementations are Ethernet, TCP/IP, or fiber infrastructure.
The control plane is used by the network to manage the topology of the data plane. The routers and switches in a network communicate over the control plane with routing and switching protocols in order to get an overview of the data plane topology. The control plane is not shown in the figure, as it can either be implemented as a small dedicated part of the data plane (in-band), or as an independent network (out-of-band).
Finally there is the management plane, which is used by the network engineers and operators to manage and monitor the network. The management plane can

use the control plane network, or a separate network. Examples of management plane protocols are Simple Network Management Protocol (SNMP) and NetConf. Examples of these applications are HP Openview[3] or Nortels DRAC[4].
Note that the management planes of different networks are shown separated in the figure above. Detailed topology or management data of different networks is not shared between them. Some limited information exchange between networks happen so far as it is required for the operation of the network, for example the announcement of prefixes and connectivity to other Autonomous Systems in case of the Border Gateway Protocol (BGP)"[5] [35].

1.4 Optical Networks and Transport Networks

An optical network is a type of data communication network built with optical fiber technology. It utilizes optical fiber cables as the primary communication medium for converting data and passing data as light pulses between sender and receiver nodes. An optical network is also known as an optical fiber network or fiber optic network.
Because of the wavelength division multiplexing (WDM) capabilities of optical networks, the term optical network was also used to describe networks that could provision dedicated network circuits. „The term lambda (after the symbol λ for wavelength), which originally meant a single wavelength on a fibre [6], started to be used for any link segment, and the term lightpath was used to refer to end-to-end circuits over these networks. Nowadays, the scope of the term has so broadened that it also includes non-optical technologies such as Ethernet" [36].
Transport networks it is networks providing circuit-switched based services, regardless of the use of optical or other transmission technologies.
„Transport networks can be created with any circuit-switched technology.
Example technologies included wavelength division multiplexing (WDM), time division multiplexing (TDM) technologies such as synchronous digital hierarchy (SDH) [7] and synchronous optical network (SONET) [8], Optical Transport Network (OTN) [9, 10], Asynchronous transfer mode (ATM) [11]. In addition, the connection oriented properties of some packet switched technologies can also be used to create virtual circuits.
Whereas a lightpath refers to a single network connection, a collection of lightpaths for the same user or organization is referred to as an optical private network (OPN) for that organization. Just like a virtual private network (VPN) is an overlay network over the regular Internet, an optical private network is an overlay network over a transport network"[36].

Chapter 2

Network Description Language

2.1 Introduction

"The routing step, which is required for provisioning of circuit switched network connections, is responsible for distributing topology information and network state across different domains. This chapter examines the distribution of topology information. It presents the Network Description Language (NDL).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 [12].
It is worth to emphasise that proposed network description language is only a method to describe topology information. It does not enforce a specific way of distributing this information, nor does it eliminate the need for a control plane for signalling and provisioning" [36].

2.2. Introduction to the Semantic Web

"The World Wide Web has allowed us to publish and share documents and information with other people in the world. However, because the web has become so popular and so widespread, it has almost become the victim of its own success. Because of the large-scale and the abundant availability of data, it becomes very hard to find what we want. Search-machines, such as Google or Yahoo, have come to the rescue and have indexed the data. However, computers still have no common sense, so the search capabilities of the search machines are rather limited. Consider for example the following two sentences:

- A is connected to B.
- There is a connection between A and B.

Even humans can differ in opinion whether these two sentences have the

same meaning. So there is no way that a computer without common sense will understand that these two lines mean the same thing. This is where the Semantic Web comes to the aid of computers (and people). The following is an excerpt of the activity statement of the Semantic Web initiative[13]:

> *The goal of the Semantic Web initiative is to create a universal medium for the exchange of data where data can be shared and processed by automated tools as well as by people. For the Web to scale, tomorrow's programs must be able to share and process data even when these programs have been designed totally independently.*

In 2000 the Semantic Web initiative was started by the World Wide Web Consortium (W3C). Since then they have been working on several specifications to publish and share (meta)data, including the Resource Description Framework (RDF) [14] and SPARQL [15]. In the following section we provide a brief introduction to RDF" [36].

2.2.1 Resource Description Framework

In order for two computer programs to communicate there must be a common understanding about the vocabulary being used. Currently most communication by computer programs is defined by protocols. The form of the interaction is fixed, but the meaning of the data being exchanged is not.
Take a web-browser for example: when a user types in a URL, the web browser starts communicating with the designated server, asking for the resource identified by the URL ('GET'). The server then answers the browser with the designated file. This interaction is strictly defined in RFC 2616[16]. But neither the web-browser nor the server know what kind of data is being exchanged, it could be about the weather, traffic information, etc.
Because in this example the applications do not grasp the meaning of the data being presented, it limits the possibilities to mere presentation. The Semantic Web idea originated as a solution to this problem; it tries to make it easier for computers to understand the meaning of the content they present, so that they can navigate autonomously through this information to find what they are looking for.
The Resource Description Framework (RDF) is a method for representing information about resources on the Web. It provides a common framework for expressing metadata so that it can be exchanged between applications without loss of meaning.
Information in RDF is expressed as statements. Each statement is a triplet, with the following elements:

- **Subject** The resource being described

- **Predicate** The property of the subject that is described
- **Object** The value of the property for the subject

A set of triplets is called a graph. An object can also be the subject of another triplet, so complex graphs can be created. An example of such a graph is shown in figure 2.1. An example triplet in the graph is 'Thesis creator John, with the subject, predicate and object respectively. The graph also contains other triplets providing more information about the object John, such as his name, email address and homepage.

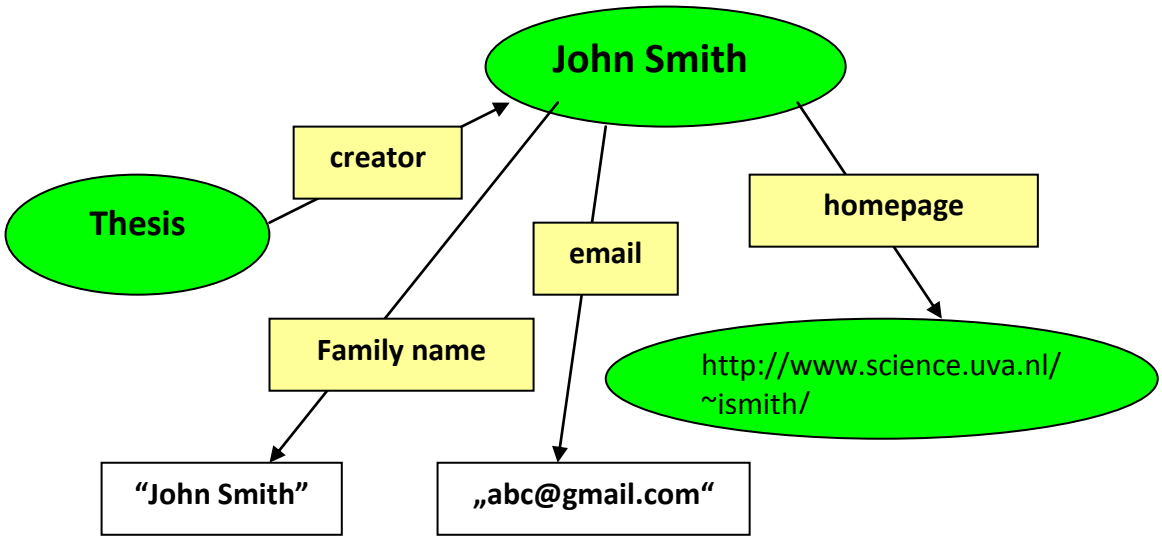


FIGURE 2.1: A SIMPLE RDF GRAPH

The graph shown in figure 2.1 still has the same problem; it is an abstract way of defining relations. RDF solves this terminology problem by using Uniform Resource Identifiers (URIs)[17]. Related terms are usually defined using the same URI-prefix, taking the form of XML namespaces [36].

2.2.2 RDF Schemata

An example of using semantics with data is the Friend of a Friend project [18]. Participants of this project describe themselves, giving their name, homepage, place of work, etc. The properties are predefined to make sure there is no conflict with e.g. using 'last name' versus 'surname'. But definitions of terms is

not enough for computers: is 'surname' the same as 'Surname'? An example of a FOAF description is given below.

```
<foaf:Person rdf:nodeID="#me">
 <dc:creator
rdf:resource="http://www.science.uva.nl/~jsmith/thesis"/>
     <foaf:family_name>John Smith</foaf:family_name>
        <foaf:mbox>abc@gmail.com</foaf:mbox>
        <foaf:homepage
rdf:resource="http://www.science.uva.nl/~jsmith/"/>
  </foaf:Person>
```

LISTING 2.1: THE RDF/XML REPRESENTATION OF THE SEMANTIC GRAPH IN FIGURE 2.1

Listing 2.1 describes the semantic graph of figure 2.1 in RDF/XML format. The properties in the example are defined using XML namespaces for readability, they actually point to specific URIs with a well-defined meaning. For example the creator property is defined by the URI ttp://purl.org/dc/elements/1.1/creator, which is defined by the the Dublin Core Initiative [19]. When defining RDF properties, it is possible to define what kind of types are valid as subject and object. The set of valid subjects is called the domain, and the set of valid objects is called the range of that property.
The other terms are either from the rdf namespace to describe standard RDF types and objects, or the foaf namespace, which provides definitions for the Person class, and basic properties of that class. Note that the homepage relationship points to a URL, but in this case it is also treated as an RDF object. This homepage object can then have other properties, such as a creator property.
An XML namespace with definitions of related terms is called an RDF schema. RDF schemata define the URIs and properties of RDF classes and RDF predicates. RDF classes define the types of subjects and objects [35].

```
<foaf:Person rdf:nodeID="#me">
     <foaf:knows>
        <foaf:Person>
            <foaf:name> John Lewis </foaf:name>
                <rdfs:seeAlso
rdf:resource="http://staff.science.uva.nl/~fdijkstr/foaf.rdf"/>
        </foaf:Person>
```

LISTING 2.2: EXAMPLE OF LINKING DESCRIPTIONS

2.2.3 RDF versus XML

There are several ways of expressing RDF graphs, one is the graphical form as in figure 2.1, and another is the statements of triplets in Notation3 in listing 2.1. The most common textual form is RDF/XML [20], where the graph is encoded in an XML format. Throughout this thesis we use the RDF/XML notation, which allows us to leverage tools for XML as well as RDF.
Besides that RDF allows reasoning about statements, it also has a few other technical advantages over other descriptions languages, such as plain XML.

Unique Identification Objects in RDF are identified by a URI. This is an advantage in multi-domain environments, since it makes it easy to clearly and uniquely define network elements in requests.

Flexible Graph Structure The relations between network elements can lead to cycles in the relation-graph. RDF extends the tree structure of XML with reference pointers so that it is able to deal with cycles.

Distributed Descriptions In order to describe inter-domain connections, the interrelation of different (operational) network domains must be described.
Each domain must be able to independently publish its own network information and point to other network domains. The RDF seeAlso predicate provides an elegant solution for this problem.

Extendable RDF schemata are easily extensible. That is, it allows users to publish all information they care about, and mix it with network schemata. The extensibility applies to both current schemata (e.g. geographic information or organizational information in geo and vcard), as well as future schemata [36].

2.3 Network Description Language

For what would to describe extensible, distributed network descriptions, developers have created a simple ontology in RDF to describe networks. The result of this work is the Network Description Language (NDL).

2.3.1 Topology Schema

The Network Description Language consists of multiple schemata, each describing a separate part of the ontology.

The topology schema of the network description language is the ontology which created to describe the topology of computer networks. An overview of the classes and properties of the topology schema is given in figure 2.2.

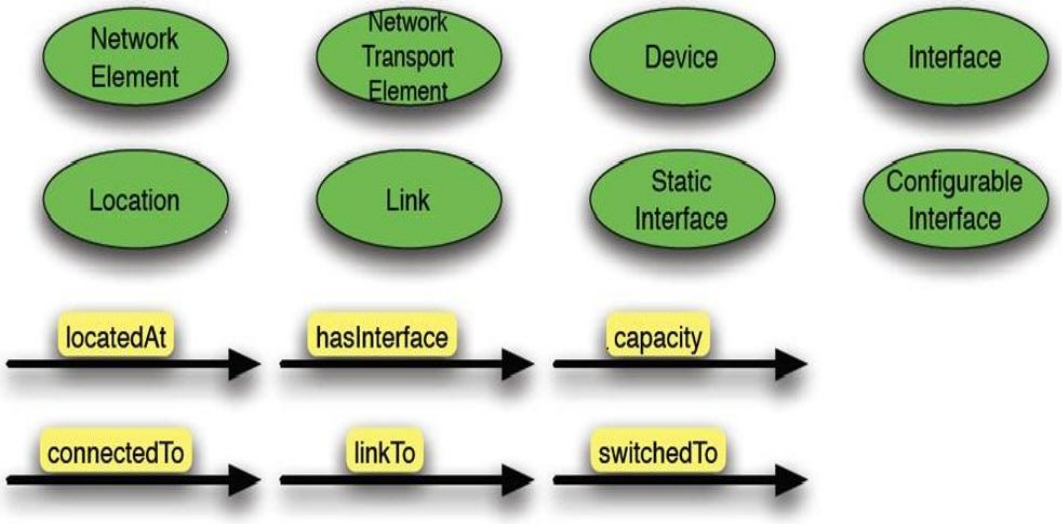


FIGURE 2.2: OVERVIEW OF THE CLASSES AND PREDICATES IN THE NDL TOPOLOGY SCHEMA

NDL has eight classes, shown at the top, that define what kind of resources can be described. The three main classes are:

Location Physical places where devices are located.

Device Devices that are part of a network.

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

NDL has six properties, shown at the bottom in the figure, to define the relations between instances of the classes and other information.

locatedAt A relation between resources and their location.

hasInterface A relation between devices and interfaces.

linkTo A relation between two interfaces, describing that they are externally connected with a direct link.

connectedTo This property is similar to linkTo, but the connection does not have to be direct; the interfaces may be connected by a series of links.

switchedTo This property is used to describe cross connects, internal connections within a device.

In addition, the predicates label from the RDF schema, and description from the Dublin Core schema can be used to describe the name and description of network elements [36].

2.3.2 Layer Schema

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 2.3.

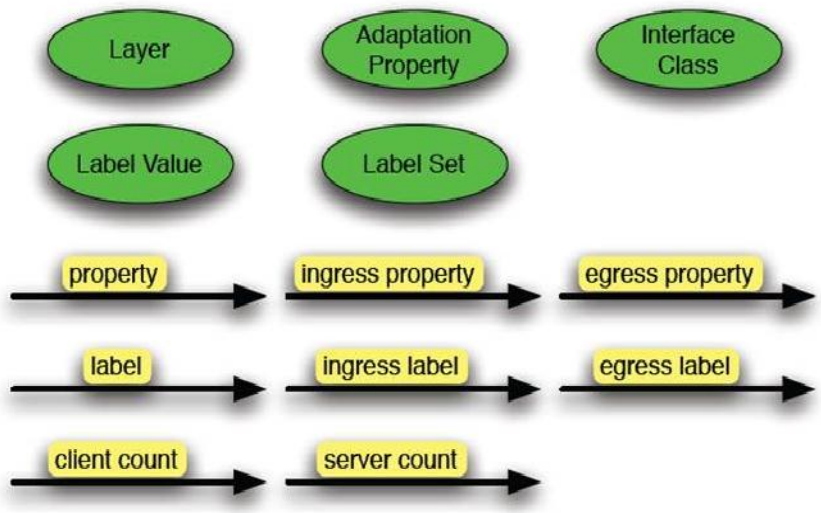


FIGURE 2.3: CLASSES AND PREDICATES IN THE NDL LAYER SCHEMA.

In figure 2.3 we show the NDL layer schema based on the G.805 model. The layer schema does not define actual adaptation functions, but instead provides 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 2.3. 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 f0; 1; 2; : : : ; 4095g.
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.

An example multi-layer description is given in listing 2.3. 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 **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, 1310nm, 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.

```
<?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#"
       xmlns:wdm="http://www.science.uva.nl/research/sne/ndl/wdm#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:layer="http://www.science.uva.nl/research/sne/ndl/layer#">
   <layer:AdaptationProperty rdf:about="http://www.science.uva.nl/research/sne/ndl
        /wdm#WDM">
      <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"
           />
      <rdfs:domain rdf:resource="http://www.science.uva.nl/research/sne/ndl/wdm#
           FiberNetworkElement"/>
      <rdfs:range rdf:resource="http://www.science.uva.nl/research/sne/ndl/wdm#
           LambdaNetworkElement"/>
      <layer:serverCount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1
           </layer:serverCount>
   </layer:AdaptationProperty>
   <rdf:Property rdf:about="http://www.science.uva.nl/research/sne/ndl/wdm#
        wavelength">
      <rdfs:subPropertyOf rdf:resource="http://www.science.uva.nl/research/sne/ndl
           /layer#label"/>
      <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
   </rdf:Property>
   <ndl:Interface rdf:about="#port3-l1310">
      <rdf:type rdf:resource="http://www.science.uva.nl/research/sne/ndl/wdm#
           LambdaNetworkElement"/>
      <wdm:wavelength rdf:datatype="http://www.w3.org/2001/XMLSchema#float">1310.0
           </wdm:wavelength>
   </ndl:Interface>
   <ndl:Interface rdf:about="#port3">
      <rdf:type rdf:resource="http://www.science.uva.nl/research/sne/ndl/wdm#
           FiberNetworkElement"/>
      <wdm:WDM rdf:resource="#port3-l1310"/>
   </ndl:Interface>
</rdf:RDF>
```

LISTING 2.3: THE NDL DESCRIPTION OF A WDM ADAPTATION [36] .

2.3.3 Domain Schema

The topology schema allows the description of physical network topologies. The NDL domain schema also allows group description of devices, links and interfaces in networks. The classes and predicates of the domain schema are shown in figure 2.4.

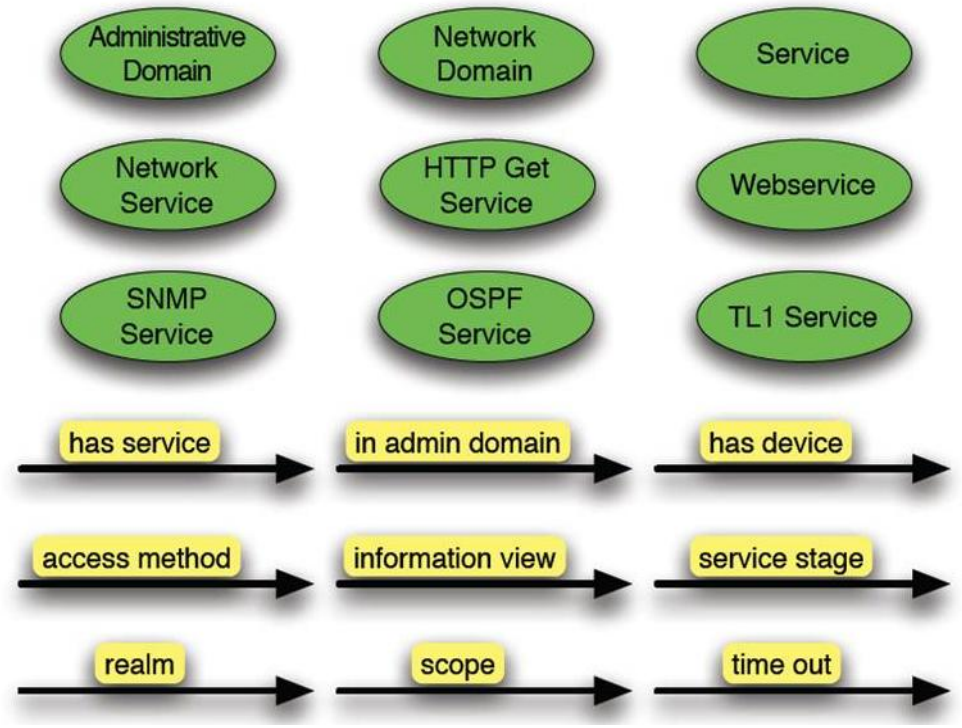


FIGURE 2.4: OVERVIEW OF THE CLASSES AND PREDICATES IN THE NETWORK DESCRIPTION LANGUAGE DOMAIN SCHEMA

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).

Using the combination of the topology and domain schema, it is possible to create descriptions of network domains. An example of such a description is shown in listing 2.4. The picture in figure 2.5 shows what is being described.

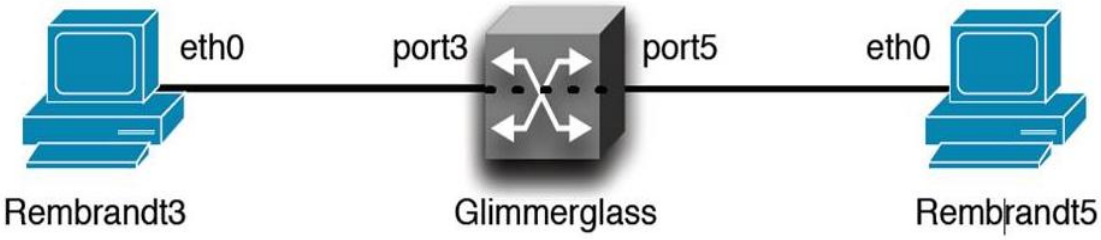


FIGURE 2.5: A SIMPLE NETWORK.

Lines 14 to 18 of listing 2.4 define the device **Rembrandt3**. **The #-prefix** on line 14 states that the device is defined in the local namespace. Line 15 provides a human readable name and line 16 states that this device is located in the location **Lighthouse** (defined on lines 11 to 13). Finally, line 17 defines that **Rembrandt3** has an interface**, Rembrandt3:eth0**. This interface is defined on lines 19 to 22. The connection to another interface is defined using the **linkTo** property on line 21, in this case it is defined to be connected to **Glimmerglass:port3**. The **Glimmerglass** device is defined similarly on lines 23–38, and the **Rembrandt5** device on lines 39–47.

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.

Our network description does not only contain a topology description, but also describes the current configuration of the **Glimmerglass** device. The **switchedTo** statement in line 32 states that the **Glimmerglass:port3** has an internal connection to **Glimmerglass:port5**.

Just like the **linkTo** property, the **switchedTo** property must be defined in both directions. So the inverse **switchedTo** property from **Glimmerglass: port5** to **Glimmerglass:port3** is also be given on line 37. With the **linkTo** and **switchedTo** statements as given above, have defined a path from the device **Rembrandt3** to **Rembrandt5** [36].

```
<?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#"

xmlns:domain="http://www.science.uva.nl/research/sne/ndl/domain#">
  <domain:NetworkDomain rdf:about="#UvALight">
    <rdf:label>UvA Light</rdf:label>
    <domain:hasDevice rdf:resource="#Rembrandt3"/>
    <domain:hasDevice rdf:resource="#Rembrandt5"/>
    <domain:hasDevice rdf:resource="#Glimmerglass"/>
  </domain:NetworkDomain>
  <ndl:Location rdf:about="#Lighthouse">
    <rdf:label>Lighthouse</rdf:label>
  </ndl:Location>
  <ndl:Device rdf:about="#Rembrandt3">
    <rdf:label>Rembrandt3</rdf:label>
    <ndl:locatedAt rdf:resource="#Lighthouse"/>
    <ndl:hasInterface rdf:resource="#Rembrandt3:eth0"/>
  </ndl:Device>
  <ndl:Interface rdf:about="#Rembrandt3:eth0">
    <rdf:label>eth0</rdf:label>
    <ndl:linkTo rdf:resource="#Glimmerglass:port3"/>
  </ndl:Interface>
  <ndl:Device rdf:about="#Glimmerglass">
    <rdf:label>Glimmerglass</rdf:label>
    <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">
    <rdf:label>port3</rdf:label>
    <ndl:linkTo rdf:resource="#Rembrandt3:eth0"/>
    <ndl:switchedTo rdf:resource="#Glimmerglass:port5"/>
  </ndl:Interface>
  <ndl:Interface rdf:about="#Glimmerglass:port5">
    <rdf:label>port5</rdf:label>
    <ndl:linkTo rdf:resource="#Rembrandt5:eth0"/>
    <ndl:switchedTo rdf:resource="#Glimmerglass:port3"/>
  </ndl:Interface>
  <ndl:Device rdf:about="#Rembrandt5">
    <rdf:label>Rembrandt5</rdf:label>
    <ndl:locatedAt rdf:resource="#Lighthouse"/>
    <ndl:hasInterface rdf:resource="#Rembrandt5:eth0"/>
  </ndl:Device>
  <ndl:Interface rdf:about="#Rembrandt5:eth0">
    <rdf:label>eth0</rdf:label>
    <ndl:linkTo rdf:resource="#Glimmerglass:port5"/>
  </ndl:Interface>
</rdf:RDF>
```

LISTING 2.4: AN EXAMPLE DESCRIPTION OF THE NETWORK OF FIGURE 2.5.

2.3.4 Distributed Repositories

In multi-domain environments there is a big potential for inconsistencies if information for each domain is not centrally maintained, and each domain replicates network descriptions of other domains. NDL addresses this issue by creating a distributed topology description, where descriptions link to each other. These links are provided using RDF's **seeAlso** statement, which points to other documents. An example of this is shown in listing 2.5**.**

```
<ndl:Interface rdf:about="#Glimmerglass:port27">
    <ndl:name> Glimmerglass:port27</ndl:name>
    <ndl:linkTo
rdf:resource="http://www.science.nl/ndl.rdf#C6509:port7"/>
  </ndl:Interface>
  <--! Test -->
  <ndl:Interface
rdf:about="http://www.science.nl/ndl.rdf#C6509:port7">
    <rdfs:seeAlso
rdf:resource="http://www.science.nl/ndl.rdf"/>
  </ndl:Interface>
```

LISTING 2.5: EXAMPLE OF DISTRIBUTED REPOSITORIES.

As shown before, lines 1 to 4 show the description of an interface of the **Glimmerglass**. However, note that in line 3, this port is defined to be connected to **http://www.netherlight.nl/ndl.rdf#C6509:port7**. On lines 5 to 7 is the definition of the interface **http://www.netherlight.nl/ndl.rdf#C6509**: **port7.** The **rdfs:seeAlso** statement is used to link to the network description of **NetherLight.** An application or crawler can then follow this pointer to the description of **NetherLight** and get more information about the interface there.
Concluding, it is possible to create a distributed network description, without a central repository [36] [35].

Chapter 3

Analyzing the problem

3.1 Introduction

Global Information Infrastructure is the technical basis for construction of the information society comprising a telecommunication, computer or may be an optical subsystem. The key task of such subsystems is to enable transmission of any type of information from any place of the world, at any time. Currently, such networks are considered as communication networks constructed according to the NGN concept. Meeting the requirements of telecommunication systems is possible by developing methods for network management as well as developing design methods.

Current telecommunication, computer systems are constructed on the principle of networks overlay which means that one transport network provides transparent transfer of data flows to another network forming its logical connections as well as its logical structure. Thus, the current telecommunication systems have a layered structure formed by the technology hierarchy.

This chapter deals with the model describing the structure of today's various systems by means of the NDL language, which takes into account their multi-level, multi-domain and multi-connection structure.
It also shows the properties and possible parameters of multi-layer graphs and proposes the describing technique, transformation, and visualizing the structure of modern telecommunication systems by means of the multi-layer graph. We also propose the search algorithm for transferring any kind of data from any place of the world, taking into account the complexity of the structure, the network parameters as well as calculation of the time spent on it.

3.2 Network modeling problems and approaches to solve them

Modern computer, optical or telecommunication networks provide ransmission of different types of traffic (information). To transmit various types of traffic over the network, its nodes must have special equipment that enables receiving incoming traffic and its subsequent transfer by outgoing communication channels. Also, the communication channels among network nodes must have sufficient bandwidth allowing transmission of information circulating within the network flows.

When constructing or upgrading such networks a network provider usually faces a problem of designing such a system. The main problem of designing is generally to determine the set of nodes and communication channels among them (network topology), to determine the capacity of communication channels and parameters of the network equipment installed in those nodes, to determine the transmission routes of data flows through the network and the number of flows transmitted via these routes.

Such networks are very complex for mathematical description. Optical (computer) networks are constructed on the hierarchical, multi-level principle, both organisationally and technologically.

Organizationally such networks are divided into levels with the higher levels managing and providing the interaction of lower levels in the hierarchy. Levels of the organization hierarchy generally have territorial separation. For example, the main network segment (WAN), the regional network (MAN) and the local area network (LAN). The levels of the technological hierarchy are overlay networks using different technologies. Each link on the upper level uses one or more ways on the lower level. For example, the IP-channel between two IP-routers can be enabled by one or more light flows in the underlying fiber-optic network.

Ideally, we can describe a multi-layer network model with the help of simple graph consisting of nodes and ribs. However, the nodes of different layers may be incompatible with each other. As a result, the final network version is not optimal, and in some cases it can cause instability in the operation of the designed network. To solve the given problem, we would like to propose using the model of multi-level network built by means of the ordered set of graphs.

Using the multi-layer graph instead of classical graphs allows consider the

technological hierarchy of modern telecommunication systems, particularly overlay principle of these systems.
The topology of each graph may differ, they can have different sets of ribs.

In the previous chapter we introduced the Network Description Language (NDL) to describe transport (computer) networks. We are going to describe the multi-layer computer or optical network with this technology-independent model and its syntax. For this purpose we extend the NDL syntax. Using an independent NDL syntax we are able to create quite a universal model and apply it to various technologies. In this case, the main goal is to do this without compromising various independent technologies.
We do this by matching the functional elements described in Chapter 2.
We are going to apply our model to various technologies, such as:

- Ethernet networks WLAN
- SONET and SDH
- WDM with a selection of wavelength switches

Another possible problem is a different of subnetwork designs that have their own parameters. These parameters are not always compatible with other network parameters. We developed a program able to distinguish all these subnetworks, their parameters and combine them into one model displayed (visualized) as a multi-layer graph. Each network or each network layer is able to contain data that may appear in other documents and which act as properties or parameters of other network. Our program is able to recognize and display such information, thereby finding and loading multiple documents at the same time.

How did we do that? Implementation of such model is carried out in RDF. This is a continuation of "Network Description Language" (NDL). In the NDL many classes and properties are organized into several modular schemes (we described them in Chapter 2). We applied three main schemes for the construction, description and visualization of such a model:

- "Topology scheme", describing the concept of devices, interfaces, and connections among them in a single layer;
- "Layer scheme", describing the concept of network layers, and relationship among the network layers;
- "Domain scheme", describing grouping of network elements in the domain, the parameters and creating an abstract representation of the network in the domain.

As it is defined in the “Topology scheme”, each interface acts as a connection point. By default each channel in each layer is presented as a single logical interface. Further we begin to describe the technical scheme for each of these technologies: IP, Ethernet, WDM, TDM (SDH / SONET). The resulting schemes describe specific technologies. The success of our model is defined by the ability to describe each of these schemes using only technology and the NDL syntax.

3.3 Encoding

Many technologies define more than one possible encoding. Encoding defines the data format for communication. For all practical goals, we define two different encodings as incompatible, and two equal encoding as compatible. Figure 3.1 displays an example of two encodings in a single technology:

Table 3.1 gives a list of a few examples of layers and encoding types. Each encoding in our program is modeled as a layer property. Such method allows us to describe the incompatibility in a single layer.

In addition to the mentioned above, we define those properties of the layer, which are not connected with different encodings. For example, we define the power level of the fiber. The advantage of the RDF is that it can be easily extended to describe such properties. It can be used for finding a particular way or technology for troubleshooting software.

Version1	**length**	**label**	**data**	**16 bit checksum**
Version2	**length**	**label**	**data**	**24 bit checksum**

FIGURE 3.1: EXAMPLE OF TWO ENCODINGS IN FOR THE SAME LAYER [35]

Layer	Incompatible Encodings
Ethernet	untagged, tagged or Q-in-Q tags
Ethernet	different maximum packet size (MTU)
DWDM spacing	100, 50 or 25 GHz spacing between wavelengths
Ethernet in UTP	10, 100, 1000 or 10000 Mbit/s

TABLE 3.1: EXAMPLES OF ENCODING TYPES FOR DIFFERENT LAYERS [35]

3.4 Problems of searching the way

As modern technology is being changed and modernized quite quickly, the problem of choosing the way is quite urgent. Not much work has been done for finding an optimal route in multi-layer networks. We know that technologies (and thus incompatibilities) change over time.

Any good algorithm for finding the way operates independently of the network; it must not be changed to take into account the specifics of a particular network. Thus, the algorithm of finding the way in the multi-layer network must be independent, universal enough and must not be modified, to take into account a small part of technology. If the algorithm for finding the way is created for a particular technology, then this algorithm must be updated or changed after updating the technology itself.

How can we cope with this problem? It should be noted that the Graphs and "Network Description Language" are independent from networks. Using them we can construct simple blocks for networks. When describing the network the instances of classes are created which are defined in the topology scheme and one or more technology schemes (such as Ethernet, WDM or TDM). Technology schemes are defined as subclasses in the "Layer scheme". Thus, in order to create an independent, universal way of finding algorithm, we have to know the scheme of the layer and the topology diagram. Based on these schemes, we will know all we need about a specific technology and will be able to find the way. You will learn more details about the search algorithm in Chapter 6.

3.5 Example of the program dealing with the network

Let us study the example of certain operations.

Let us study the system consisting of five physical devices. Each of these devices is composed of multiple domains and has its own interface, which may differ from interfaces of other devices.

These devices use the IP-network operating over the Ethernet network, and uses an optical signal transmission cable. The given network has the following layers in its structure:
- The layer of IP-network, which is represented by a collection of IP-nodes connected by IP-channels;
- The layer of Ethernet network, represented by a number of nodes interconnected by communication channels acting as transport environment for the IP-network layer;
- The optical network layer represented by the nodes of optical network connected with optical communication cables. In the nodes of optical network optical splitters, multiplexers/demultiplexers and other optical network equipment can be installed;
- the layer of cable channels represented by cable channels applied for laying fiber optic cable.

We can see the multi-layer network scheme in Figure 3.2

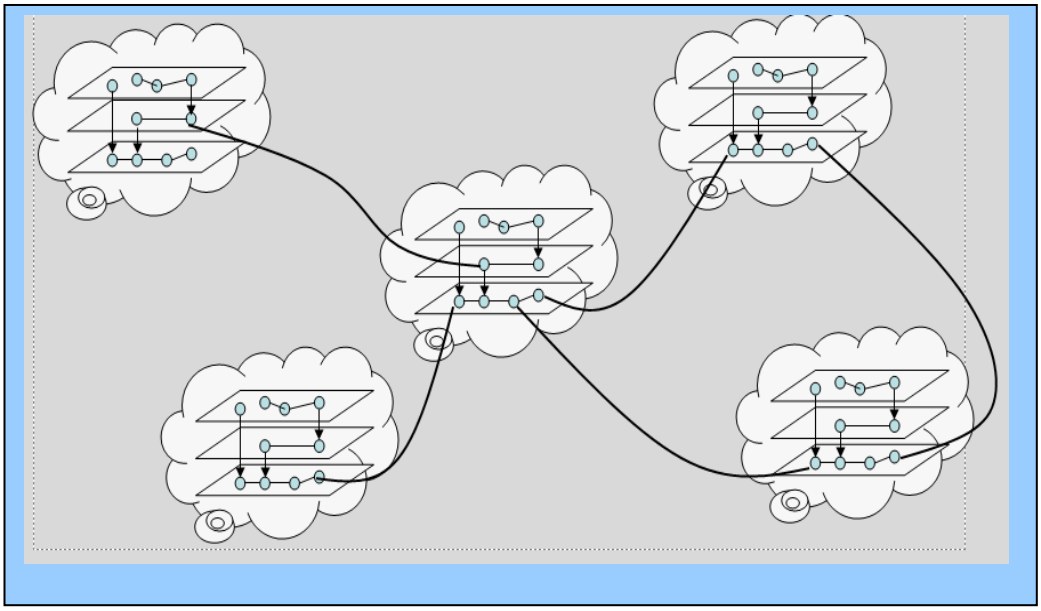


FIGURE 3.2: SCHEME OF MULTI-LAYER NETWORK

For describing such a network with a multi-layer graph we perform the

following operations:
1.1. Select a number of layers in the modelled telecommunications system.
1.2. Describe the topology of each layer and domain by means of the NDL syntax.
1.3. Describe each layer topology by means of the classical graph.
1.4. Distinguish among objects of different layers logical, functional and physical links and visualize (draw) them using graphs.
1.5. Assign a set of parameters, characterizing the corresponding objects and inter-object links to ribs and vertices of the graph.

The example of describing the network topology using the NDL syntax can be seen in Listing 3.1.

```
<ndl:Device rdf:about="#Lls001a_ome01">
  <ndl:name>Lls001a_ome01</ndl:name>
  <ndl:locatedAt rdf:resource="#SURFnet6"/>
  <ndl:hasInterface rdf:resource="#Lls001a_ome01:9/1"/>
</ndl:Device>

<ndl:Interface rdf:about="#Lls001a_ome01:9/1">
  <ndl:name>Lls001a_ome01:9/1</ndl:name>
  <ndl:connectedTo rdf:resource="#Zl001a_ome01:5/1"/>
  <ndl:capacity rdf:resource="#OC192"/>
  <ndl:encapsulation rdf:resource="#Ethernet "/>
</ndl:Interface>

<ndl:Device rdf:about="#Zl001a_ome01">
  <ndl:name>Zl001a_ome01</ndl:name>
  <ndl:locatedAt rdf:resource="#SURFnet6"/>
  <ndl:hasInterface rdf:resource="#Zl001a_ome01:1/1"/>
  <ndl:hasInterface rdf:resource="#Zl001a_ome01:5/1"/>
  <ndl:hasInterface
rdf:resource="#http://www.netherlight.nl/ndl.rdf#C6509:port7"/>
</ndl:Device>

<ndl:Interface rdf:about="#Zl001a_ome01:5/1">
  <ndl:name>Zl001a_ome01:5/1</ndl:name>
  <ndl:connectedTo rdf:resource="#Lls001a_ome01:9/1"/>
  <ndl:capacity rdf:resource="#OC192"/>
  <ndl:encapsulation rdf:resource="#SONET"/>
  <ndl:Interface
rdf:about="http://www.netherlight.nl/ndl.rdf#C6509:port7">
    <rdf:seeAlso
rdf:resource=”http://www.netherlight.nl/ndl.rdf”/>

  </ndl:Interface>
</ndl:Interface>
```

LISTING 3.1: THE NDL DESCRIPTION OF A MULTI-LAYER AND MULTI-DOMAIN NETWORK

When the program performs the operation 1.3, for each separate layer, selected

in step 1.1 and 1.2 the network topology is determined. For this purpose:

2.1. At the given layer each node of the network is replaced with a vertex of the graph.
2.2. Pairs of interacting nodes are determined.
2.3. For each pair of interacting nodes we introduce a rib connecting the corresponding vertices.

For the upper layer corresponding to the level of IP-network, the graph will contain a set of vertices representing the nodes of the IP-network. The graph corresponding to the IP-network layer is a set of fully-connected subgraphs. The layer graph corresponding to the layer of Ethernet network, contains vertices corresponding to the nodes of the Ethernet network. In the networks constructed with use Ethernet technology, nodes can:
- interact directly with each other (in the case of use in general transmission environment)
- interact through the central node (in the case of dial-up connections).

Depending on this the topology of the graph of this layer is different. Thus in both cases, the graph of this layer contains many subgraphs whose number corresponds to the number of Ethernet segments of the modelled network. In this case the connection between subgraphs is performed by the rib connecting vertices of subgraphs matching the network nodes where a switch or a bridge connecting Ethernet segments of the network is installed.

The graph of the layer level of the optical network contains a number of vertices standing for the nodes of the optical network, where optical splitters, multiplexers/demultiplexers and other equipment are installed. The ribs of the graph stand for optical cables connecting the nodes of the network and thus the graph topology repeats the optical network topology.

While performing operation 1.4 the multi-layer graph gets ribs connetcting vertices which belong to different layers. Each introduced rib stands for the logical and physical link between nodes of the overlay networks.

While performing operation 1.5 each rib of the multi-layer graph receives a number of parameters corresponding to the parameters of the modelled network. The following parameters act as such parameters for the network in question.
For the layer graphs corresponding to the overlay networks:
- channel capacity between the network nodes;

- message processing time in the node;
- parameters of data flow transmitted through communication channels.

Such graph visualization is displayed in Figure 3.3. The possible parameters are shown in Figure 3.4. The results of searching the way taking into account the parameters are shown in Figure 3.5.

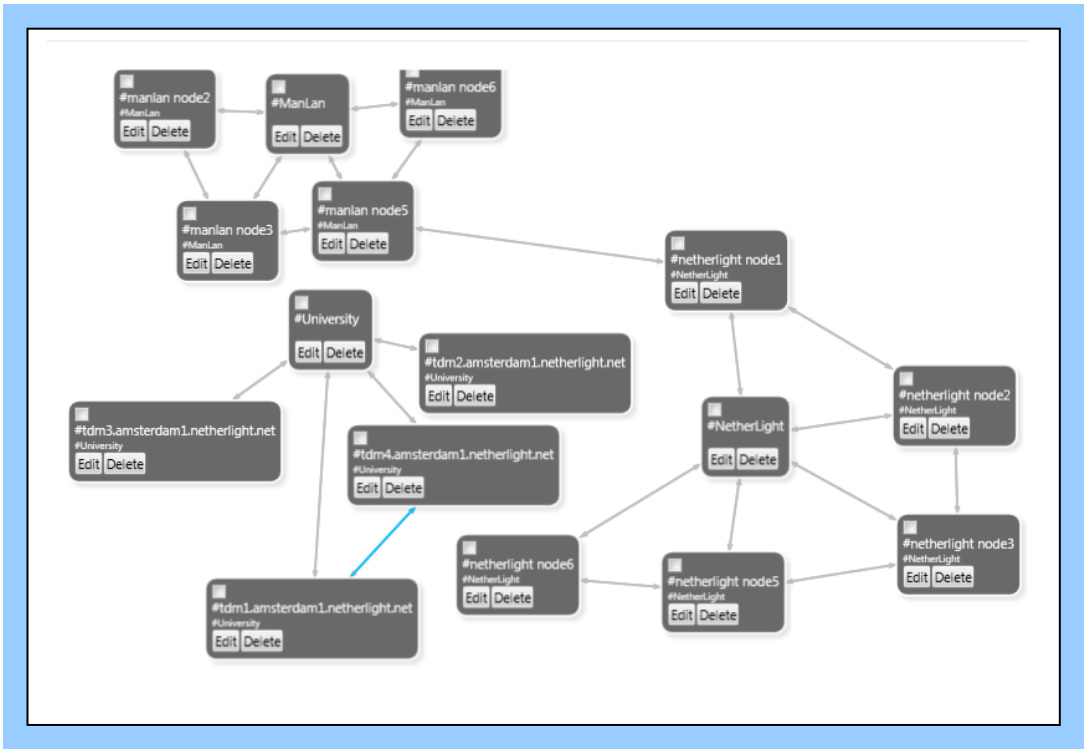


FIGURE 3.3: VISUALIZATION OF A MULTI-LAYER AND MULTI-DOMAIN NETWORK

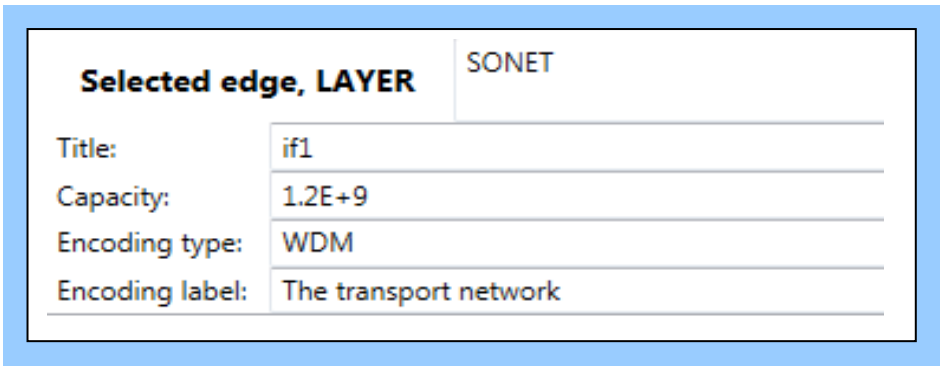


FIGURE 3.4: THE POSSIBLE PARAMETERS

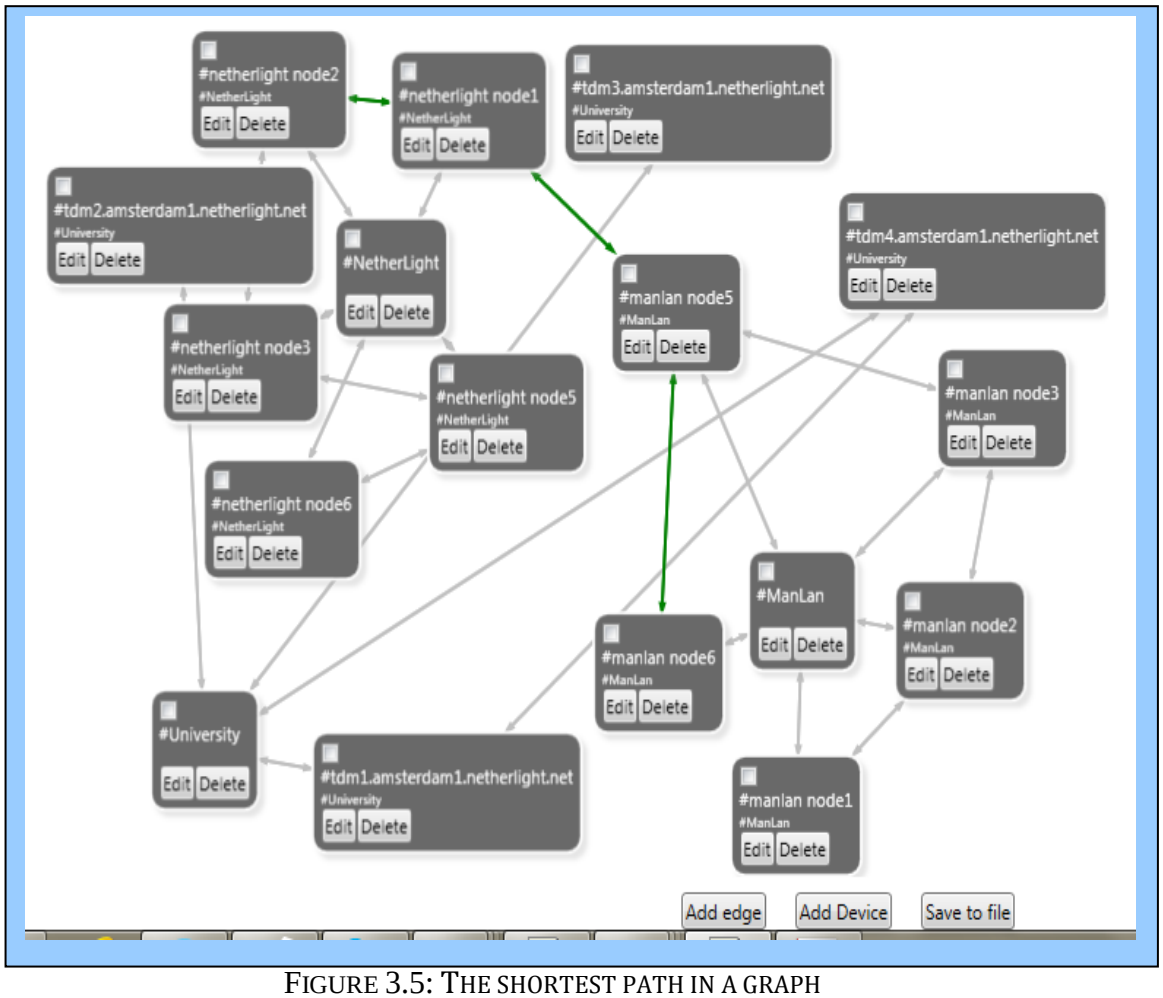


FIGURE 3.5: THE SHORTEST PATH IN A GRAPH

Chapter 4

NDL Applications

4.1 Introduction

In the previous chapter we have introduced the Network Description Language. I have shown an example of how it can describe devices and link between different domains.
"One of the advantages of using NDL as the language for description of hybrid networks is the availability of semantic web tools for RDF that can parse and consume the information in each NDL file. This means that extracting of the information needed for network management is straightforward and simple.
In this chapter we show example of how the language can be applied for solving the operational issues that operators and users face in hybrid optical networks and other kinds of networks.
Network operators and users often use maps of the network to make sense of the topology and be able to provide support in diagnostics or manual path finding". [A2]
Primarily we need to gather data. After developing application one can see that it is helpful to have an available toolkit in order to support the most common tasks performed with NDL files.

4.2 Applications

Main goals of NDL applications application can be listed by the following points:

- Reading and extracting Data from Network Descriptions
- NDL data visualization.
- Modification of the visualized structure.

- NDL Graph creation and editing.
- Storing of the edited documents.
- Finding of the shortest path in NDL graph

4.2.1 Reading and extracting Data from Network Descriptions

As we described in the previous chapter NDL is based on RDF/XML. This has several advantages, one of which is that we can make use of generic RDF/XML tools and standards. It applies a simple syntax to specify variables and triplet templates for retrieving information from a repository.

4.2.2 Visualization using RDF/XML tools.

One of the advantages of using NDL as a language for description of hybrid networks is the availability of semantic web tools for RDF, which can parse and consume the information in each NDL file. This means that extracting of the information needed for network management is straightforward and simple.
In my work a visualization tool is implemented. It takes network description in NDL format in order, to get the connections between the devices and their names. This data is then converted in to the graph by using the scripts.

4.2.3 Modifications of the Visualized Structure

RdfVisualizer enables editing of each every element, its properties, like name or namespace and its attributes. It is possible to add or remove elements from the document thus allowing modifications of the tree hierarchy.

4.2.4 NDL Graph creation and editing.

Program enables editing of every node and its ports. It is possible to add or remove nodes/ports from the document thus allowing modifications of the tree hierarchy.
New document can be created in a variety of formats.

4.2.5 Storing of the edited documents.

NDL documents can be saved back to the text format, exported to various image formats and sent to the printer.

4.2.6 Finding the shortest path in a NDL graph

The application allows the user to choose the vertices and each vertex has a label. The distance between the vertices in fact is measurement for shortest-path calculations. The distance is calculated automatically by calculating the distance between the vertices. A report with the total distance and path found will be shown [33] .

Chapter 5

User Guide

5.1 User Interface

The Work Project offers tools for data visualization of NDL documents, generation of NDL schema document and finding of the shortest path in an NDL graph.
The following sections describe in details “what” and “how” you can achieve.

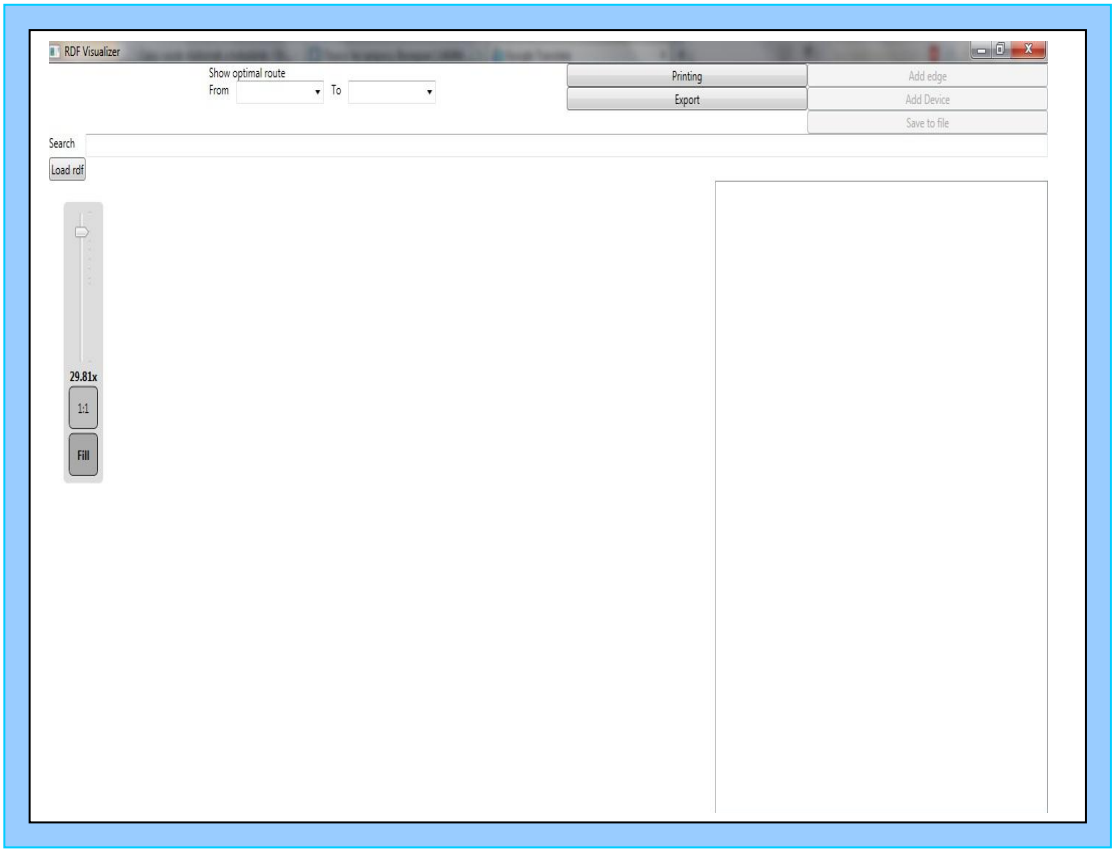


FIGURE 5.1: MAIN WINDOW

Figure 5.1 shows the main window of the program. It contains the following, parts:

- Loading document – here you can upload your document
- View NDL structure – window view the structure of your document.
- Canvas – the main working area.
- Zoom – enables to zoom-in or zoom-out the canvas.
- Select nodes – here you can select the start and end point for the search path.

5.2 View NDL structure

First you need to upload your document. You need to click on the "Load". This situation is shown in figure 5.2

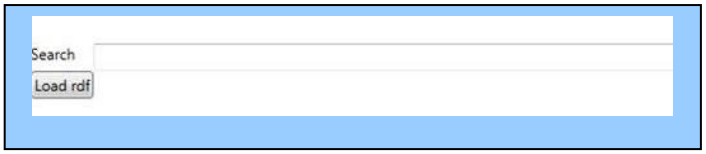


FIGURE 5.2: LOADING DOCUMENT

After selecting the file in the left part of the window you will see the hierarchical structure of the NDL document. Tree view of the NDL document, containing nodes and paths, is displayed. It is also possible to expand, the collapsed node. All this is shown in figure 5.3 and figure 5.4.

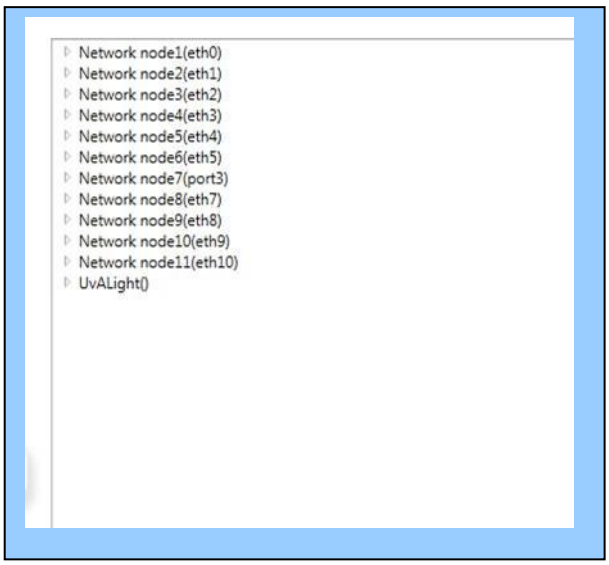


FIGURE 5.3: HIERARCHICAL STRUCTURE

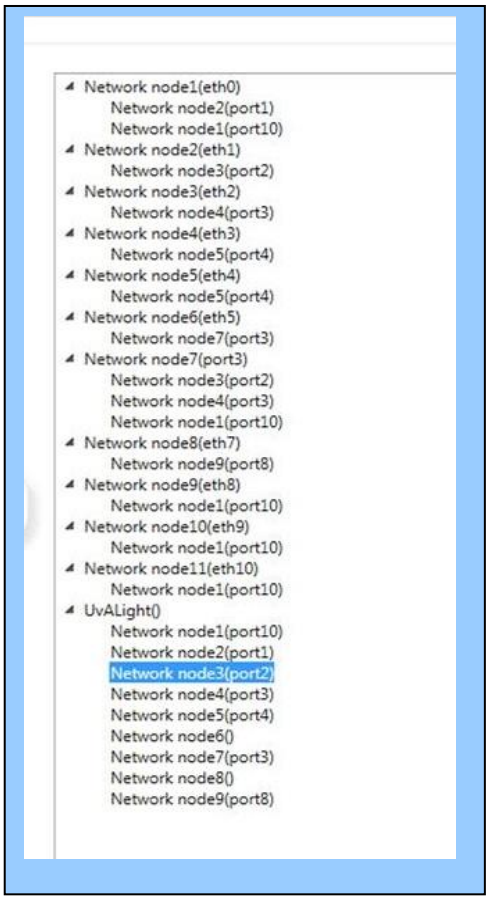


FIGURE 5.4: EXPANDING NODES

5.3 NDL data visualization (the NDL topology schema)

In order to see a graphical representation, first it is necessary to load the document. The program will draw your graph, according to the document scheme, in the right part of the screen. This is shown in figure 5.5

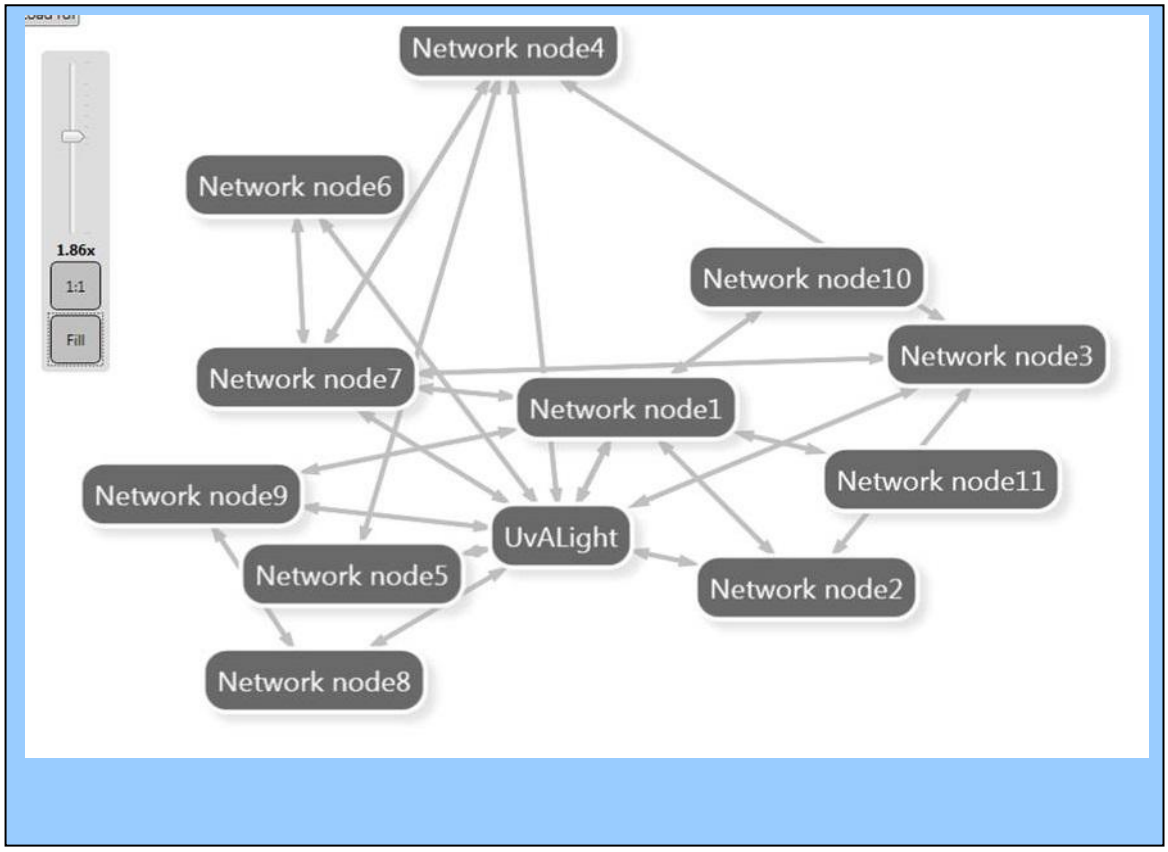


FIGURE 5.5: DOCUMENT VISUALIZATION

Figure 5.6 shows the one of child elements on canvas. The child element is connected to its brothers with a line ended with an arrow. Relationships between elements can be both unilateral and bilateral. Bilateral ties are red pointers, but one-way links between elements, it is blue pointers. These links can be seen by hovering the mouse over the node.

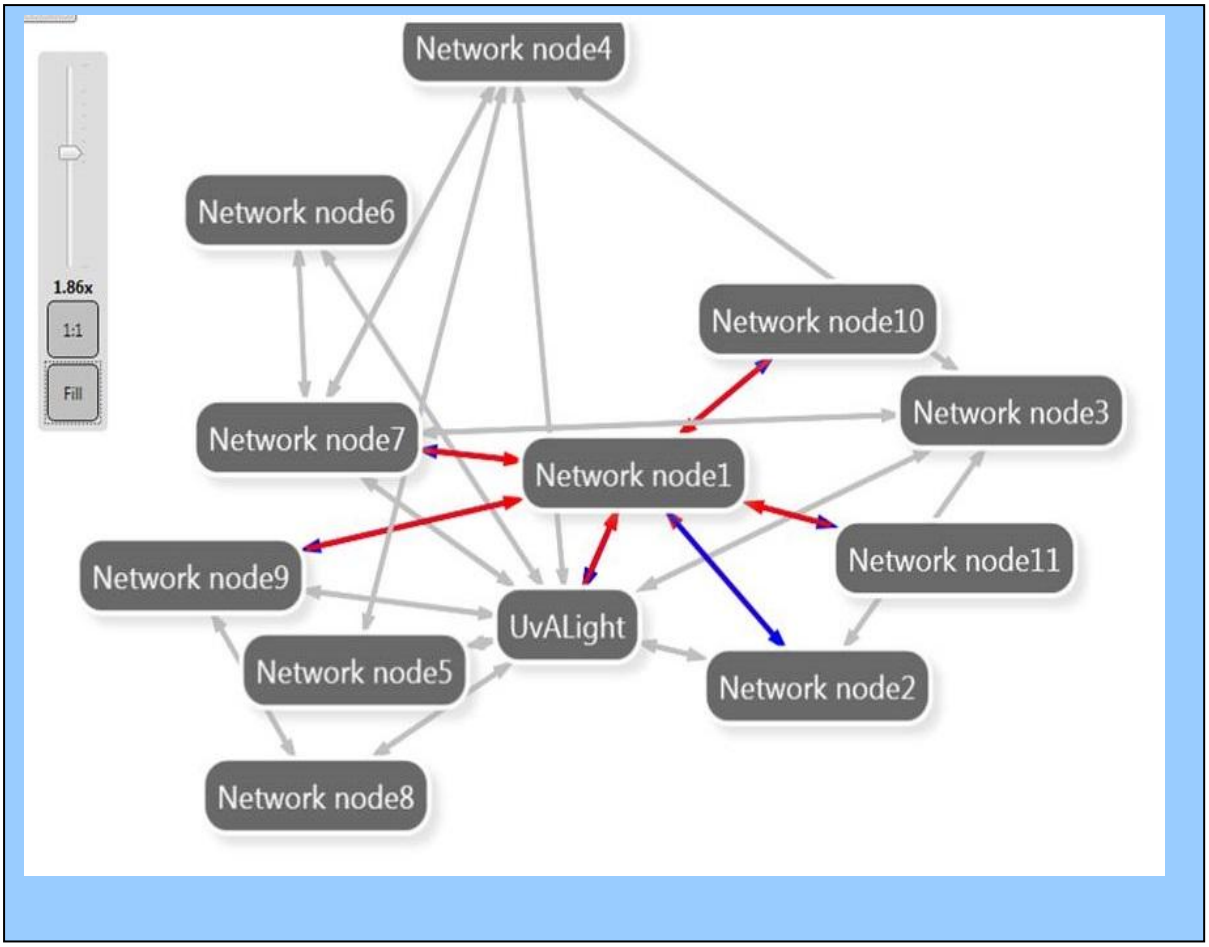


FIGURE 5.6: RELATIONSHIPS BETWEEN ELEMENTS

Every visualized item has its port (the definition of relationships between elements) (figure 5.7) which visible when the mouse is over it.

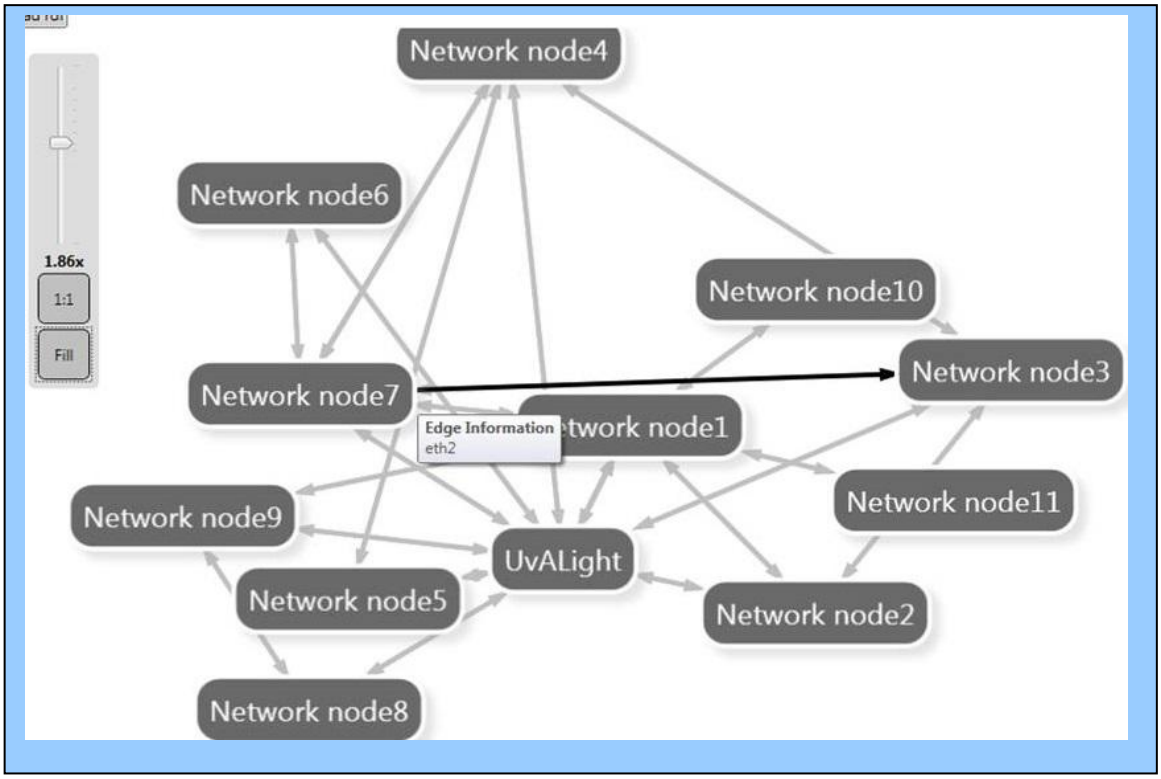


FIGURE 5.7: THE DEFINITION OF RELATIONSHIPS

5.4 Finding the shortest path in an NDL graph

For finding the shortest path in an NDL graph you need to select nodes in the loaded document. Select nodes according to your query, as shown in figure 5.8. The nodes selected by the query will get label "selected". The shortest path between specified elements will be displayed in the window.

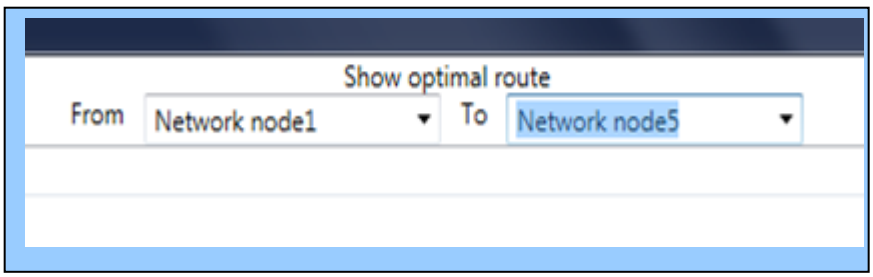


FIGURE 5.8: THE SELECTED NODES

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. As a result, the shortest path in the graph will be shown a green arrow. Figure 5.9

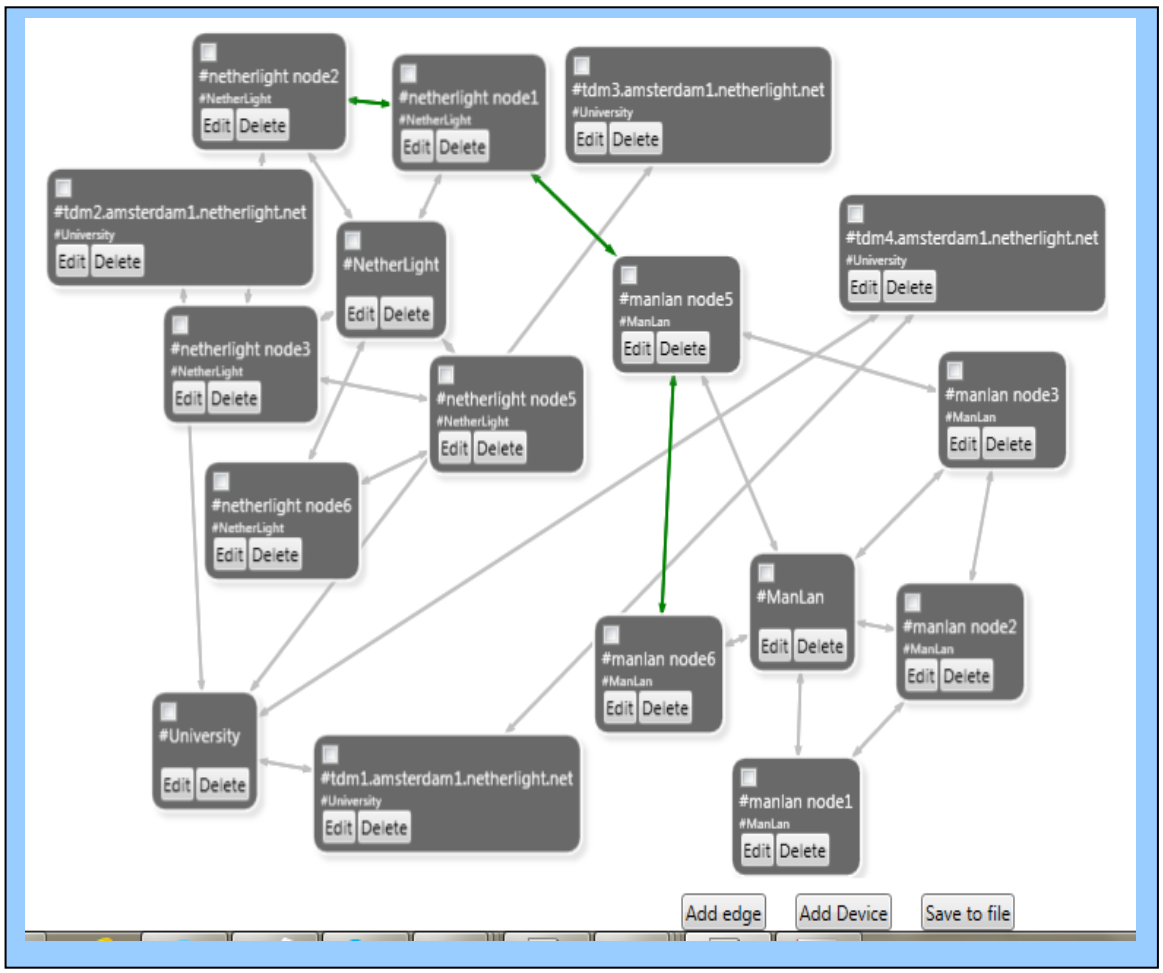


FIGURE 5.9: THE SHORTEST PATH IN A GRAPH

5.5 Creation and manipulation of graphs

For adding new nodes/edges to a graph, we have to click the button "Add Device"/"Add edge". We add in a new window a new node name and the name of resource.

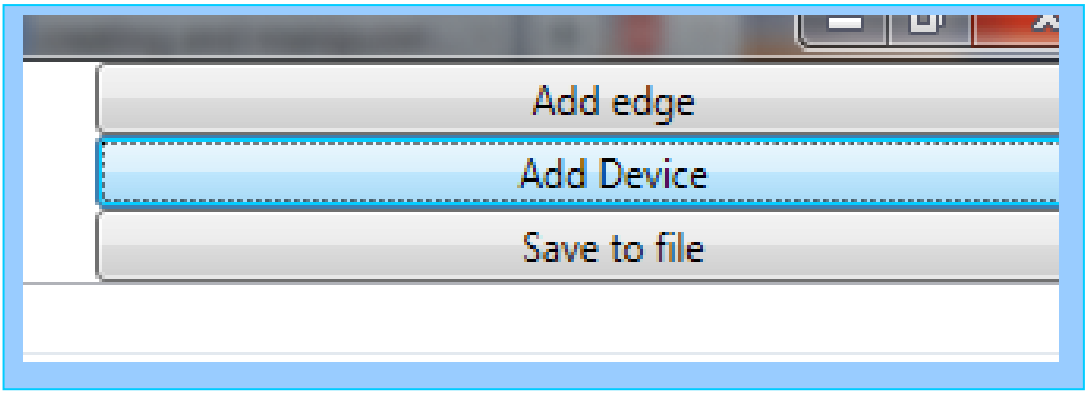


FIGURE 5.10: ADDING NEW NODES/EDGES TO A GRAPH

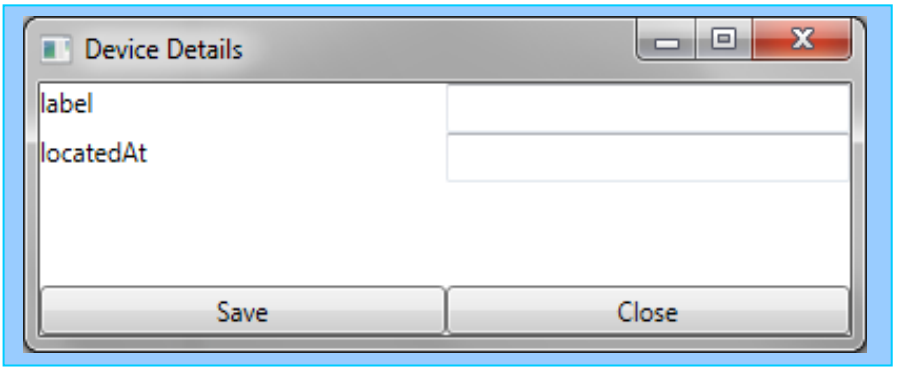


FIGURE 5.11: ADDING THE NODE NAME

For editing or deleting of the existing node, we have to allocate the necessary nodes, and then click, the appropriate button. After that the button "Save to file" and the whole document will be saved to the file. Figure 5.12.

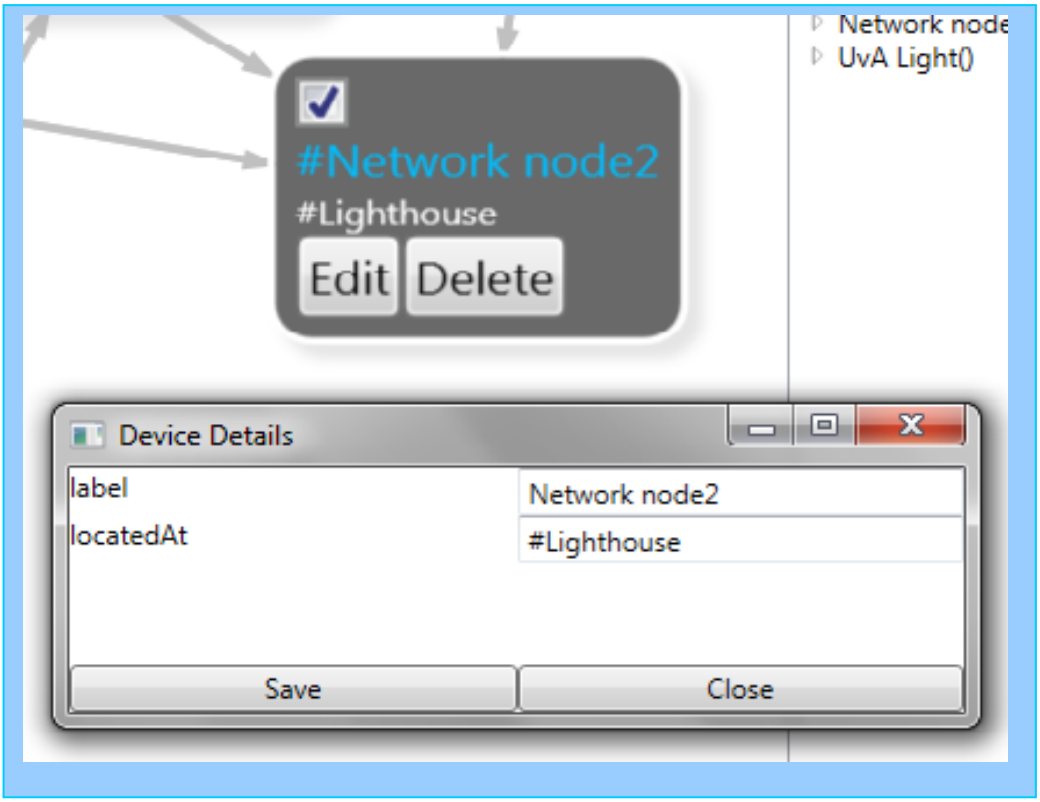


FIGURE 5.12: EDIT/DELETE OF THE EXISTING NODE

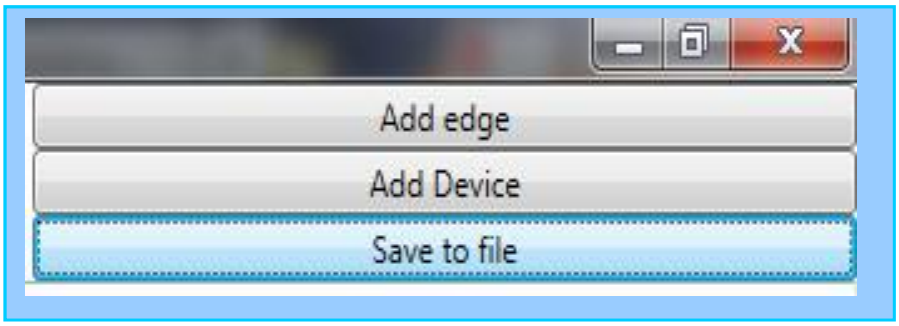


FIGURE 5.13: SAVE TO FILE

5.5.1 Export

The graph of NDL document can be saved to one of the following formats: XML, RDF, PNG. To export the current graph, click on the "Export" button.

5.5.2 Printing

The visualized NDL document can also be sent to the printer. To do so, click on the "Print" button.

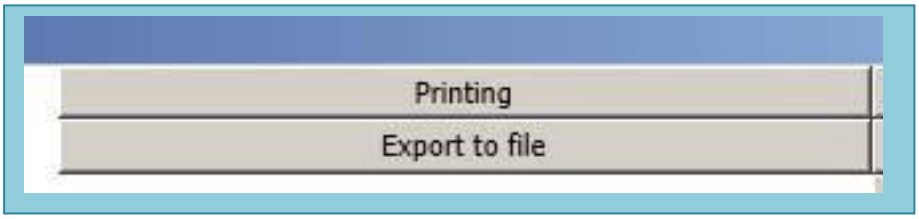


FIGURE 5.14: EXPORT AND PRINTING

Chapter 6

Shortest-paths problem

6.1 Introduction

In graph theory, the shortest path problem given weighted, directed graph G = (V, E), positive weight function w : E → R. The weight of a path $p = (V_0, V_i, ..., V_k)$ is the sum of the weights of its constituent edges.
Weight of the shortest path from vertex u to vertex v is defined by

$$\delta = \begin{cases} \min\{w(p) : u \xrightarrow{p} v, if \quad \exists \quad a \quad path \\ \infty, \quad otherwise \end{cases}$$

Then, by definition, the shortest path from vertex u to vertex v - it is the way, the weight of which satisfies $w(p) = \delta(u, v)$.

The weight of each of edges can be interpreted as other metrics. Often they are used for representation of time intervals or any other value which needs to be minimized.
This section presents the Bellman-Ford algorithm that allows to solve the problem of the shortest path from a fixed node in the general case, when the weight of any edge can be negative. This algorithm is characterized by its simplicity. Its advantages also include the fact that it determines whether the cycle in a graph with negative weight, reachable from the node.
Also in the section describes the algorithm of Dijkstra which is characterized by smaller time of performance, comparing to the Bellman-Ford algorithm, but requires non-negative weight of each of the ribs was.
Furthermore, an algorithm for dynamic programming - Floyd-Warshall algorithm, which allows to solve the problem of finding the shortest paths between all pairs of nodes, is included in the section. [21]

6.2 Bellman–Ford algorithm

The Bellman-Ford algorithm finds the distance from one vertex (assigned number 0) to all other vertices, where the weight of each of the edge can be negative. For a directed weighted graph G = (V, E) with root s and weight function w: E $\rightarrow$R Bellman-Ford algorithm returns a Boolean value indicating whether the graph contains a cycle with negative weight from the root. If such a cycle exists, the algorithm indicates that no solutions exist. If there are no cycles, the algorithm shows shortcuts and their weight.

Asymptotic complexity:

- Average case (random data): O(| V |·| E |)
- Worst case: O(| V |·| E |)

Formal description

```
Bellman_Ford(G, w, s)
1:  INITIALIZE (G,s)
2:  for i ← 1 to |V|-1
3:      do for each edge (u,v)∈E
4:          do RELAX (u,v,w)
5:  for each edge (u,v)∈E
6:      do if d[v]>d[u]+w(u,v)
7:          then return FALSE
8:  return TRUE
```

LISTING 6.1: THE BELLMAN–FORD ALGORITHM [22]

6.3 Dijkstra's algorithm

Dijkstra's algorithm finds the shortest distance from one of the vertices of the graph to all others. All weights must be non-negative.

Running time.

The simplest implementation of the Dijkstra's algorithm stores vertices of set Q in an ordinary linked list or array and extract minimum from Q-it is simply a linear search through all vertices in Q. In this case, the running time is $O(|E| + |V^2|) = O(|V^2|)$.

Formal description

Dijkstra (G, w, s **)**

1: **INITIALIZE_SINGLE_SOURCE (G,s)**

2: $S \leftarrow 0$

3: $Q \leftarrow V[G]$

4: **While** $Q \neq 0$

5: **do** $u \leftarrow Extract\_Min\ (Q)$

6: $S \leftarrow S \cup \{u\}$

7: **for** each edge $v \in Ad_j[u]$

8: do RELAX (*u,v,w)*

LISTING 6.2: DIJKSTRA'S ALGORITHM

Example. The algorithm of Dijkstra's: figure 6.1

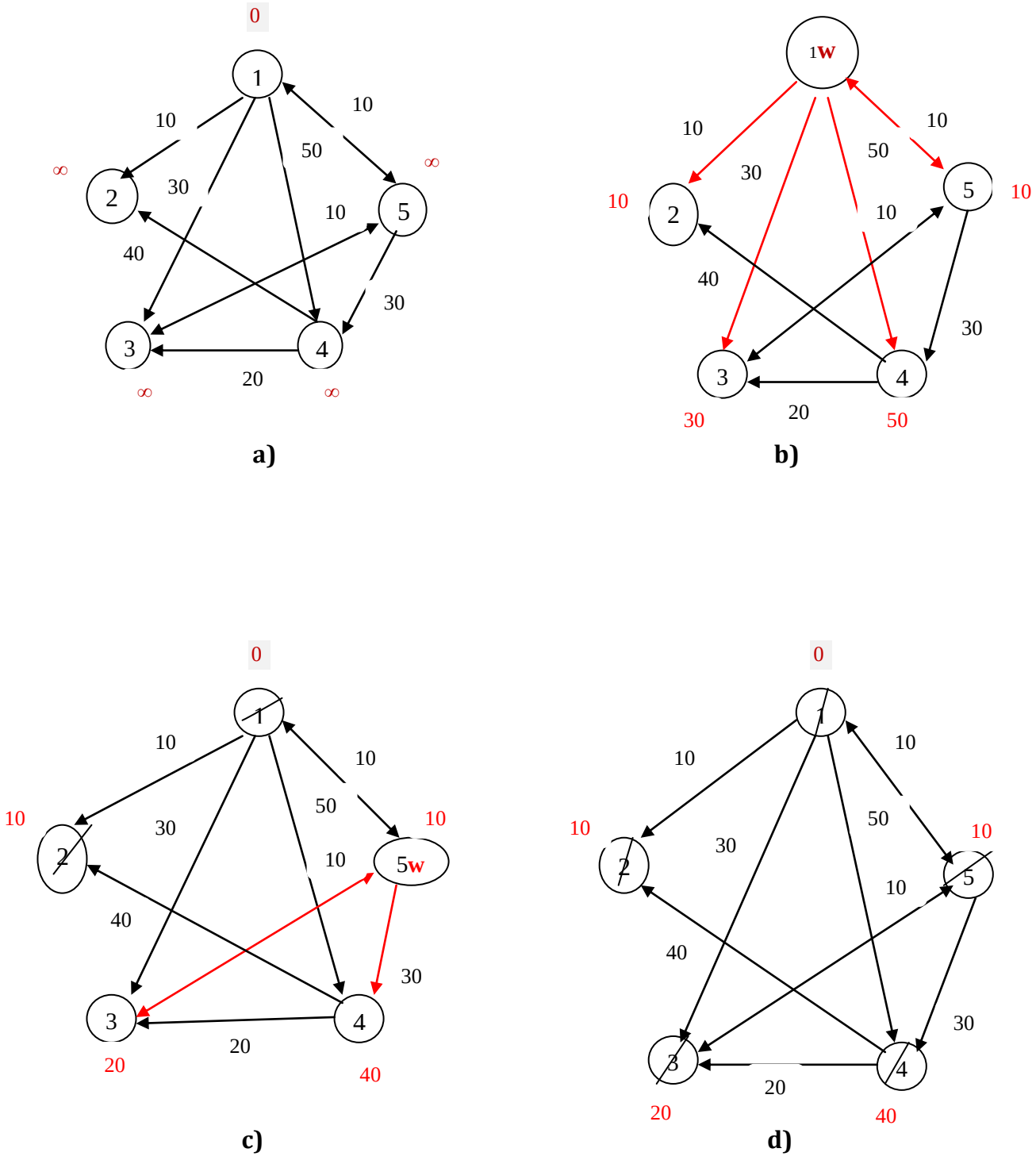


FIGURE 6.1: DIJKSTRA'S ALGORITHM [23]

6.4 Floyd–Warshall algorithm

The Floyd-Warshall algorithm-dynamic algorithm for finding the shortest distances between all vertices of a directed weighted graph.
Defined directed graph $G = (V, E)$ each arc $v \longrightarrow w$ of the graph corresponds to a non-negative value of $C [v, w]$. The general problem of finding of the shortest paths is to find for each ordered pair of vertices *(v, w)* of any path from vertex *v* to vertex *w*, whose length is minimal among all possible paths from v to *w*.

Formal description

Let **A** be a $N \times N$ matrix (***N*** is the number of vertices), **A[i, j]** representing the length (or cost) of the shortest path from *i* to *j***.**
"For each element **A[i, j]** assign a value equal to the cost of the edge going from *i* to *j*, or an infinity value if this edge doesn't exist.

At each step, for each pair of vertices *i* and *j* see if there's an intermediate vertex *k* so that the path from *i* to *j* through *k* is shorter than the one already found for *i* and *j***.** If ***i, j*** and *k* are ordered properly, only $O(N^3)$ operations are needed to find the values of all elements of **A**. Such an order is obtained when first *k* is considered and then *i* and *j*".[24]
After the *k*-th iteration, **A[i, j]** contains the value of the smallest path lengths from vertex *i* to vertex *j*, which does not pass through the vertices with index greater than k. At the *k-th* iteration to compute the matrix *A* formula is used:

$$\mathbf{A_k[i,j]=min(A_{k-1}[i,j],\ A_{k-1}[i,k]+A_{k-1}[k,j])}$$

To calculate $\mathbf{A_k[i, j]}$, value $\mathbf{A_{k-1}[i, j]}$ is compared with a value:

$$\mathbf{A_{k-1}[i, k]} + \mathbf{A_{k-1}[k, j]}$$

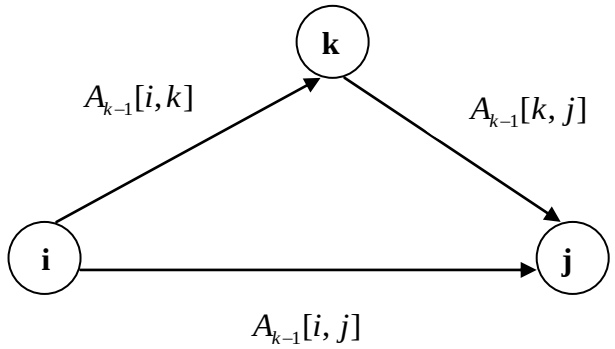
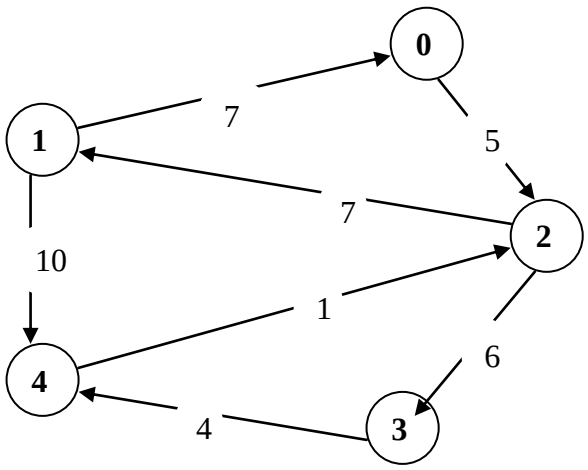


FIGURE 6.2: INCLUSION THE VERTEX K IN THE PATH FROM THE VERTEX I TO THE VERTEX J[25]

Example. The algorithm of Floyd-Warshall: figure 6.3



	0	1	2	3	4
0	0	12	15	----	22
1	7	0	11	----	10
2	14	7	0	----	17
3	19	12	5	0	4
4	15	8	1	----	0

FIGURE 6.3: EXAMPLE OF A GRAPH AND TABLE OF DISTANCES

6.5 Experience of application and conclusions

After detailed analyses of the considered methods for creation of optimum ways. I can conclude that from all listed algorithms, the most convenient and effective algorithm of the solution of a task about finding of the shortest way is Dijkstra's algorithm. The advantage of this algorithm over other methods is that for sufficiently large number of vertices, the computing time is less. The algorithm of Dijkstra's was applied in my work.

6.6 Multi-layer network model

In this section, we provide three network descriptions. The first model is a model in which each device represents node in a graph G, and network links are represented as edges. The second model is a model in which each network device as multiple nodes in a graph Gl. The third model where transform the multi-layer network into a graph Gs consisting of nodes and links on different ‘‘technologies”.

6.6.1 Device-based network description G

Figure 6.4 is example of a multi-layer network. This network we provide six devices, D={A,B,C,D,E,F}, here we consider two layers L={Ethernet, SONET(STS)}. There two layers of the adaptation: Gigabit Ethernet (GE) can be compatible in 24 STS channels or in 21 STS channels(STS-3c-7v where is it 7 virtual containers of 3 concatenated channels).
It is designated as A(STS) = { 24c, 3c7v }. The network has 6 physical links, and not all devices support all adaptations. In this network the shortest correct path from A to C is A–B–E–D–B–E–F–C, because then GE is adapted in STS-24c at node B and 22 STS channels are available between B and D.
The graph G in figure 5.4 is a primary way to description the physical properties of a network, with devices as nodes, and links as edges. Information on (de)adaptation capabilities is present in the nodes, in the graph defined by G ={N,E}.

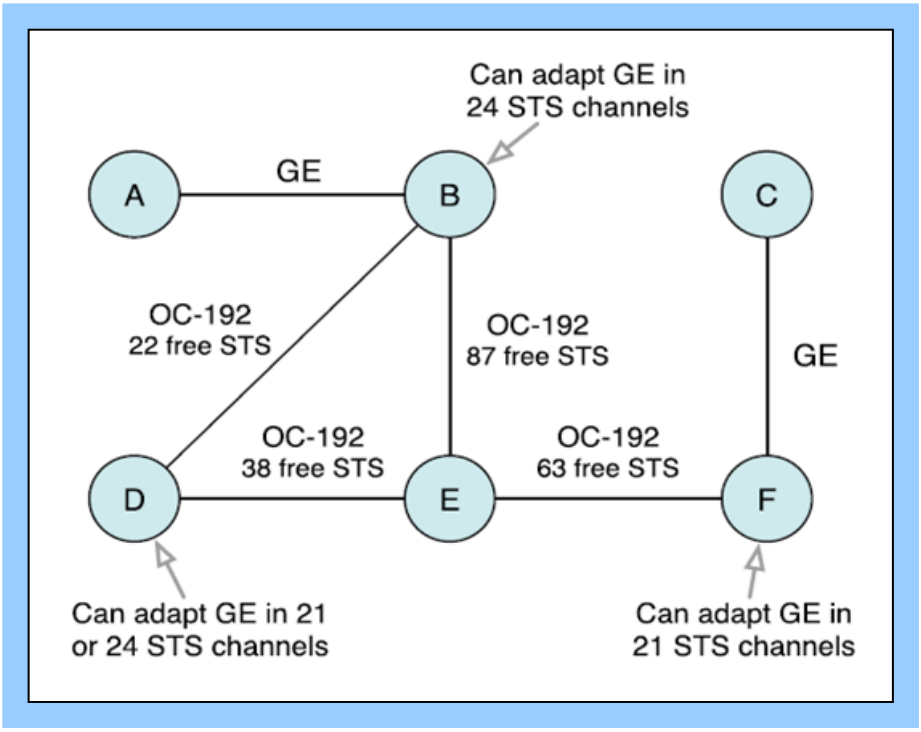


FIGURE 6.4: EXAMPLE OF A MULTI-LAYER AND MULTI-DOMAIN NETWORK [36]

6.6.2 Stack-based network description Gs

For creation of effective algorithm of the count ***Gs*** we must come to a simple description of the network, which consists of nodes and edges. (De)adaptations from one technology to another are represented as simply by vertical edges. Consider the example network in figure 5.4. We will define three adaptation ***: S = {Ethernet, Ethernet over 24 STS channels, Ethernet over 21 STS channels}.***
We will create the graph ***Gs=(Ns,Es)*** of a multi-layer network. We will assume that the network consists of D devices and S technology stacks, in this case the graph contains of ***Ns = S * D*** nodes. Nodes on the same row have an edge, if they can communicate with each other directly. An edge from a node in one row to a node in another row represents a (de)adaptation. "By assigning weights to the horizontal edges we can represent the cost of using an edge, and by assigning weights to the vertical edges we can represent the cost of (de)adaptationIn figure 6.5, we presented more detailed scheme of the network from figure 6.4 (we use edge weights for the link capacity)". [37]
Node B can only use GE in 24 channels, node ***D*** can use ***GE*** into 24 channels or into 21 channels. Nodes on the same row ***t*** can communicate among themselves without (de)adaptations. The dotted line, shows that these nodes should be able to communicate among themselves.

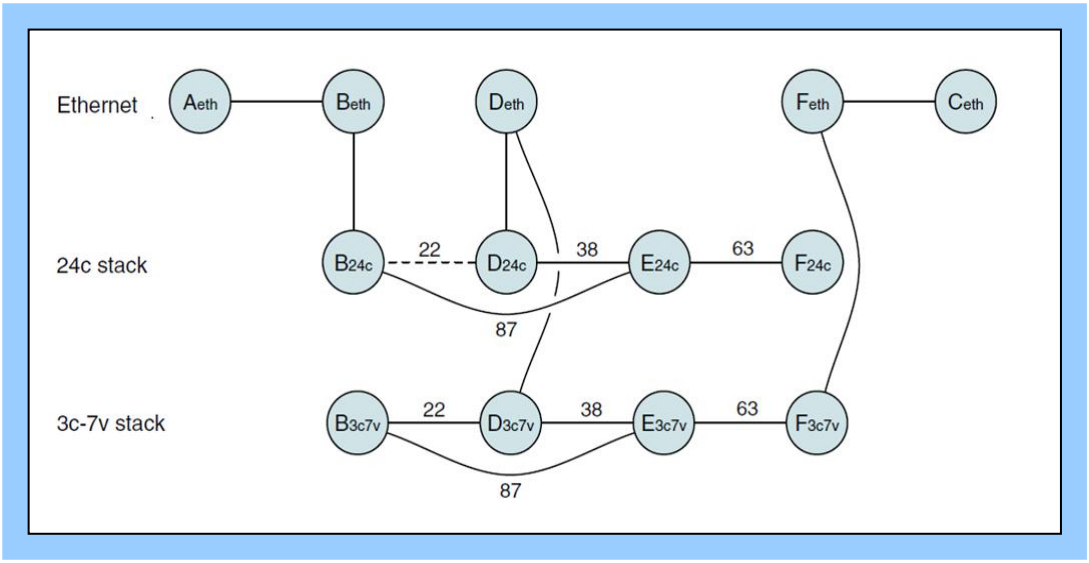


FIGURE 6.5: REPRESENTATION OF THE NETWORK IN Figure 6.4 AS A MULTI-LAYERED GRAPH [36]

6.6.3 Path selection in *Gs*

We will consider the Dijkstra algorithm to the graph ***G*** in figure 6.4 from node A in node ***C***, in this case the shortest the path it is A–B–E–F–C. If we use the Dijkstra algorithm to the graph ***Gs*** in figure 5.5, we would find the path **A**Eth–**B**Eth–**B**24c–**E**24c–**D**24c–DEth–**D**3c7v–**E**3c7v–**F**3c7v–**F**Eth–**C**Eth, which will correspond to path A–B–E–D–E–F–C.

For finding of an optimum path (A–B–E–D–B–E–F–C), we need to create algorithm, taking into account capacity on physical links traversed multiple times. We need to check for enough bandwidth. If is insufficient bandwidth, then subsections of shortest paths are not optimum (not necessarily shortest). It is algorithm for path selection in ***Gs*** is MULTI-LAYER-DIJKSTRA (Gs, s, t,B). The algoritm of MULTI-LAYER-DIJKSTRA(Gs, s, t, B) is given below.

```
1. DIJKSTRA(Gs, t) → r[v]/*Lower bounds for all nodes*/
2.  for all nodes v ∈ Ns
3.     d[v] ← ∞ /*The distance to s*/
4.     π [v] ← NIL/*The predecessor node to s*/
5.     counter[v] ← 0
6.  maxlength ← ∞
7.  d[s] ← 0
8.  counter[s] ← counter[s] + 1
9.  INSERT(Q,s, counter[s], NIL, d[s] + r[s])
10. while Q ≠ 0;
11.     EXTRACT-MIN(Q) → u[i]
12.     if u[i] = t
13.        return path
14.     else
15.        for each v ∈ adj[u[i]]/*for each neighbor v of u[i]*/
16.           if d[u[i]] + w(u[i],v) + r[v]< maxlength
17.              if FEASIBLE (u[i],v,B)/*Backtracking*/
18.                 d[v] ← d[u[i]] + w(u[i],v)
19.                 p[v] ← u[i]
20.                 counter[v] ← counter[v] + 1
21.                 INSERT (Q,v,counter [v],u[i], d[v] + r[v])
22.                  if v = t
23.                     maxlength ← d[v] + r[v]
24.  x ← u
25.  y ← i
26.  B‘ ← b(u,v)
27.  while( π [x[y]] ≠ NIL)
28.        if L( π [x[y]],x) = L(u[i],v)
29.           B‘ ← B‘- B
30.        x[y] ← π [x[y]]
31.        if x = v
32.        return FALSE
33.  if B‘ < B
34.        return FALSE
35.  else
36.        return TRUE
```

LISTING 6.3: MULTI-LAYER-DIJKSTRA ALGORITHM

We need to find a shortest path from ***s*** to ***t*** in ***Gs***, which has bandwidth of ***B***.
" ***d[v]*** gives the shortest found distance from ***s*** to ***v***.

π ***[v]*** gives the predecessor of node ***v*** that was used to reach node ***v*** with distance ***d[v].***
counter[v] refers to the number of paths stored at node ***v*** and ***maxlength*** refers to the maximum length that a (sub)path may have.
w(u,v) and ***b(u,v)*** give the weight and the available bandwidth on the edge ***(u,v),*** respectively." [37]
We define ***c(e)*** to be the available capacity of the physical link ***e. L(u,v)*** defines the relation of a given edge ***(u,v)*** to the corresponding physical link. We created nodes ***n*** as ***n(d, s)*** for device ***d*** and technology stack ***s, L(u,v) = L(u(du, su),v(dv, sv))*** is the physical link between devices ***du*** and ***dv***.
We define ***b(u,v) = c(L(u,v)). C*** is the set of capacities for all physical links in the network.
B is the required capacity of the path, which may also differ per stack.
Line 1 computes the shortest path from ***t*** to all other nodes in the graph using the classical Dijkstra algorithm and does not consider any restrictions. For a start we need to describe all nodes (lines 1–9).
The weights of these paths, referred to as ***r[v]*** for all nodes ***v*** $\in$***Ns***. If the shortest path from ***s*** to ***t*** will be found, we can finish the algorithm and return result. In other case the algorithm will proceed.
The weights of these paths is it lower bound estimates, ***r[v]*** for all nodes ***v*** $\in$ ***Ns***.
Line 9 inserts the source node with length ***d[s] + r[s] = r[s]*** and predecessor ***NIL*** in the queue ***Q***.
Line 11: the node ***u[i]*** from the queue has the shortest weight. Since multiple paths can be stored at a node ***u***, ***u[i]*** is used to denote the ***ith*** path at node ***u***. Not entire paths are stored, but by backtracking the predecessor list p, the entire path can be reconstructed. Node ***u[i]*** is the new scanning node towards destination ***t***. If ***u[i] = t*** we have found the shortest path.
„**Line 16** checks whether the length of the path extended from ***u[i]*** to v does not exceed ***maxlength***, otherwise it is discarded because we already have a better candidate. If this first test is passed, we continue by backtracing the predecessor list to check, with the module FEASIBLE(), for loops and for enough available bandwidth on the physical link ***(u,v)***. If these tests are also passed, we may insert v into the queue.“ [37]
The complexity of ***MULTI-LAYER-DIJKSTRA(Gs, s, t,B)***. **Lines 1–9** have the complexity ***O(Ns*logNs + Es). kmax*** the maximum number of paths stored at each node. If node can be selected at ***kmax*** times from the queue, the ***EXTRACT-MIN*** function in ***Line 11*** takes value ***O(kmax*Ns*log (kmaxNs)).***
Total complexity for MULTI-LAYER-DIJKSTRA ***O(kmaxNslog kmaxNs) + kmaxNsEs***), where ***kmax*** may be exponential in ***Ns***.

6.6.4. Creating the search algorithm

Above we have described search algorithms for multi-layer graphs.

All these algorithms are based on Dijkstra's algorithm. The above mentioned

algorithms are quite interesting and complex enough. They include the parameters and limits that can contain optical (telecommunication) networks. But in our opinion, these algorithms do not apply the significant parameters that are very important when dealing with this kind of technologies. If we do not take into account the way finding algorithm it may not work properly and it cannot be optimal, that is very important when operating with optical technologies.

We developed the algorithm which modifies the classical Dijkstra's algorithm and adds parameters and limits which are important in our opinion.
Above all we would like to add a parameter called switch. We have already told that in our graph the nodes represent either devices or network interfaces, and the ribs stand for the communication channels or adaptation between network layers. As a rule, the switches may stand both at the input and output of a device. That means that when passing through a certain interface or an adaptation device parameters of communication channels may change, to put it simply we see a process of switching. We must take into account that some switches already have certain connections that we are not able to change, because it may lead to an error or degrade already existing communications. In this case, our algorithm contains two options:

1. If such adaptation is suitable, the "algorithm runs on";
2. If such adaptation is incompatible with our adaptation, the algorithm starts to look for another way using other communication channels.

The second important parameter is the wavelength (lambda). This is quite an important parameter to be considered when creating the search algorithm. The wavelength parameter affects the speed and has a great importance when calculating the propagation time of the algorithm.

The third significant parameter added to our algorithm is the power ("capacity"). The problem is that the power of devices is not always given in their description or given by certain encodings, for example: "AS-12.01" or "B09." Our algorithm can solve this problem in the following manner. If the parameter of the device capacity is not specified, it is determined by default, i.e., we use a certain standard value. If the capacity parameter is encoded, the program translates this code into the numeric format. Thus, we receive a certain number which is taken into account when calculating the execution time of the algorithm.

The fourth parameter, which we added to our algorithm is a type of encoding ("encoding type"). As long as we have a multi-layer network which contains several types of encoding, we need to check the adaptation of various encodings. If encodings are not compatible, the algorithm will have to look for other ways, spending extra time.

So we explained how the created algorithm operates. We are sure that this algorithm is very simple to use. We tried to make it easy to understand for users. This algorithm can surely be modified, complicated or simplified. Also the parameters (limits) we added may be removed.

Certainly, considering the fact that the optical (communication) technology is rapidly changing, improving, so parameters for search algorithm will change. That is why we tried to make our work as accessible as possible for understanding and further modification.

Implementation of our algorithm can be seen in Chapter 8.

Chapter 7

Architecture of the project

7.1 Introduction

The architecture of my project consists of several main parts. I consider that such approach helps to keep all integrity of my work and it is easier to understand the project for users and for programmers. I tried to simplify my work.

7.2 Overview of the Architecture

In my work I applied the C# language using NET Framework 4.5 as a development platform. The application consists of three main parts. This is shown in figure 7.1. The overall architecture is inspired by the WPF: MVVM (Model View View-Model) [26].
„The Model-View-View-Model (MVVM) is an architectural pattern used in software engineering that originated from Microsoft which is specialized in the Presentation Model design pattern. It is based on the Model-view-controller pattern (MVC) and is targeted at modern UI development platforms (WPF and Silverlight) in which there is UX developer who has different requirements than more "traditional" developer. MVVM is a way of creating client applications that leverages core features of the WPF platform, allows for simple unit testing of application functionality, and helps developers and designers work together with less technical difficulties“ [27].
Give a simple definition of Model View View-Model.

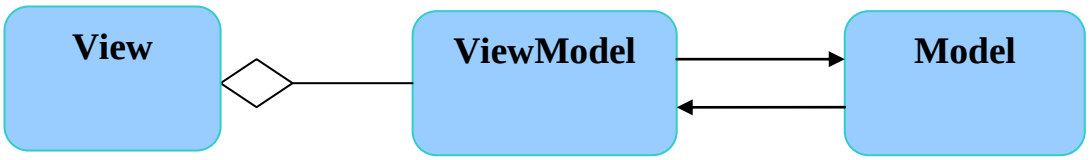


FIGURE 7.1: EXAMPLE OF MVVM

Figure 7.1 shows overview of program. It contains the following parts:

1. The View holds a reference to the ViewModel. The View basically displays stuff by Binding to entities in the View Model.
2. The ViewModel exposes Commands, Notifiable Properties, and Observable Collections to the View. The View Binds to these ViewModel entities/members
3. The Model is your data and/or application objects that move data while applying Application Logic. If you have a Business Layer, then you might not need this.

The ViewModel is the most significant in the entire pattern as it is the glue that sits between the View and the Model and binds both of them together[26].

7.2.1 The View

A View is defined in XAML. The View can contain the visual controls, animations and other functions of the navigation for the purpose of the visual representation. The View is data bound to the Model. The View acts as communications between your software and users.

7.2.2 The Model

Model is responsible for exposing data in a way that is WPF. Model can implement INotifyPropertyChanged and/or INotifyCollectionChanged. When data is expensive to fetch, it abstracts away the expensive operations, never blocking the UI thread.

7.2.3 The ViewModel

A ViewModel is a model for a view in the application. The ViewModel exposes data and exposes the behaviors for the view with Command. The ViewModel is the glue between the View and the outside environment. It is what the View is bound to. It provides a specialization of the Model that the View can use for data-binding.

7.3 Architecture of decision system

RdfVisualizer is an application which contains common Models, Views and ViewModels. The most important classes and interfaces are shown in figure 7.2.

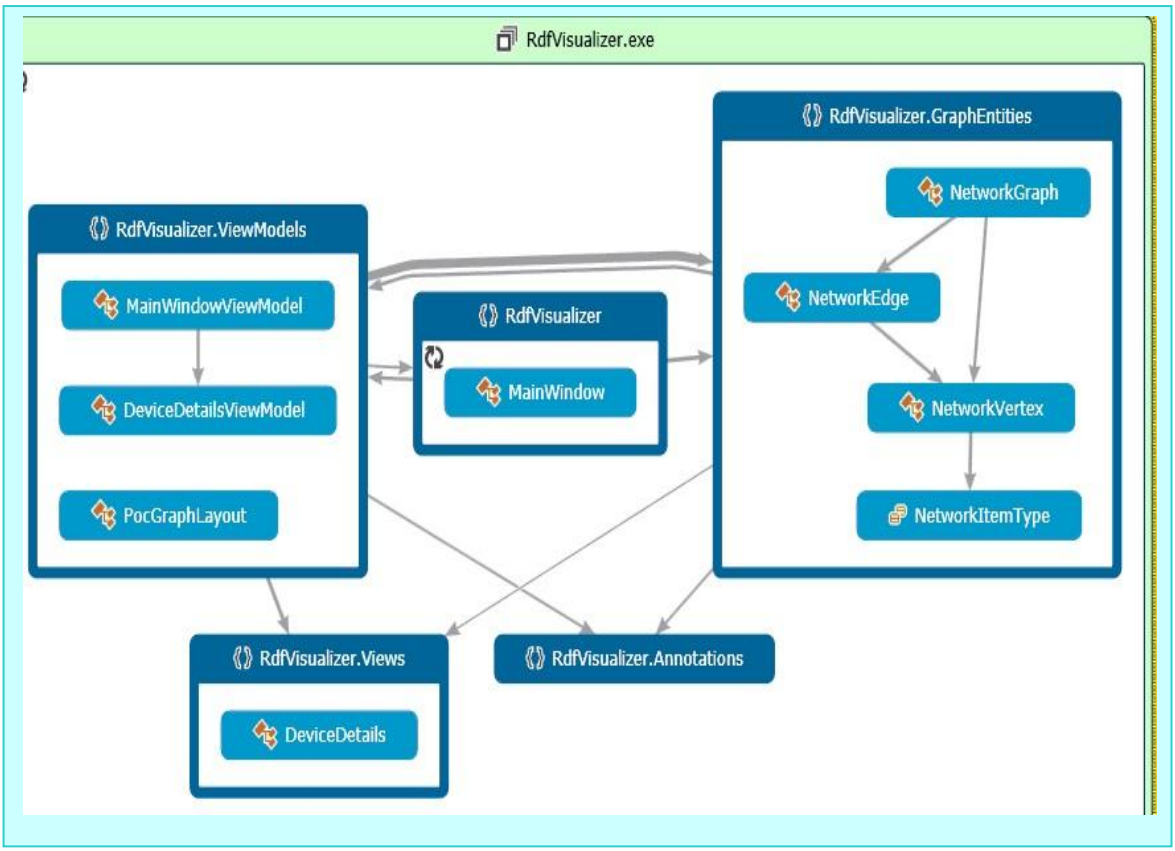


FIGURE 7.2: BASE MODELS AND VIEWS

7.4 Models and helper classes

GraphEntities contains realization of the main models and other helper classes which should be implemented for the description and visualization of the graph of the initial document. This can be seen in figure 7.3.

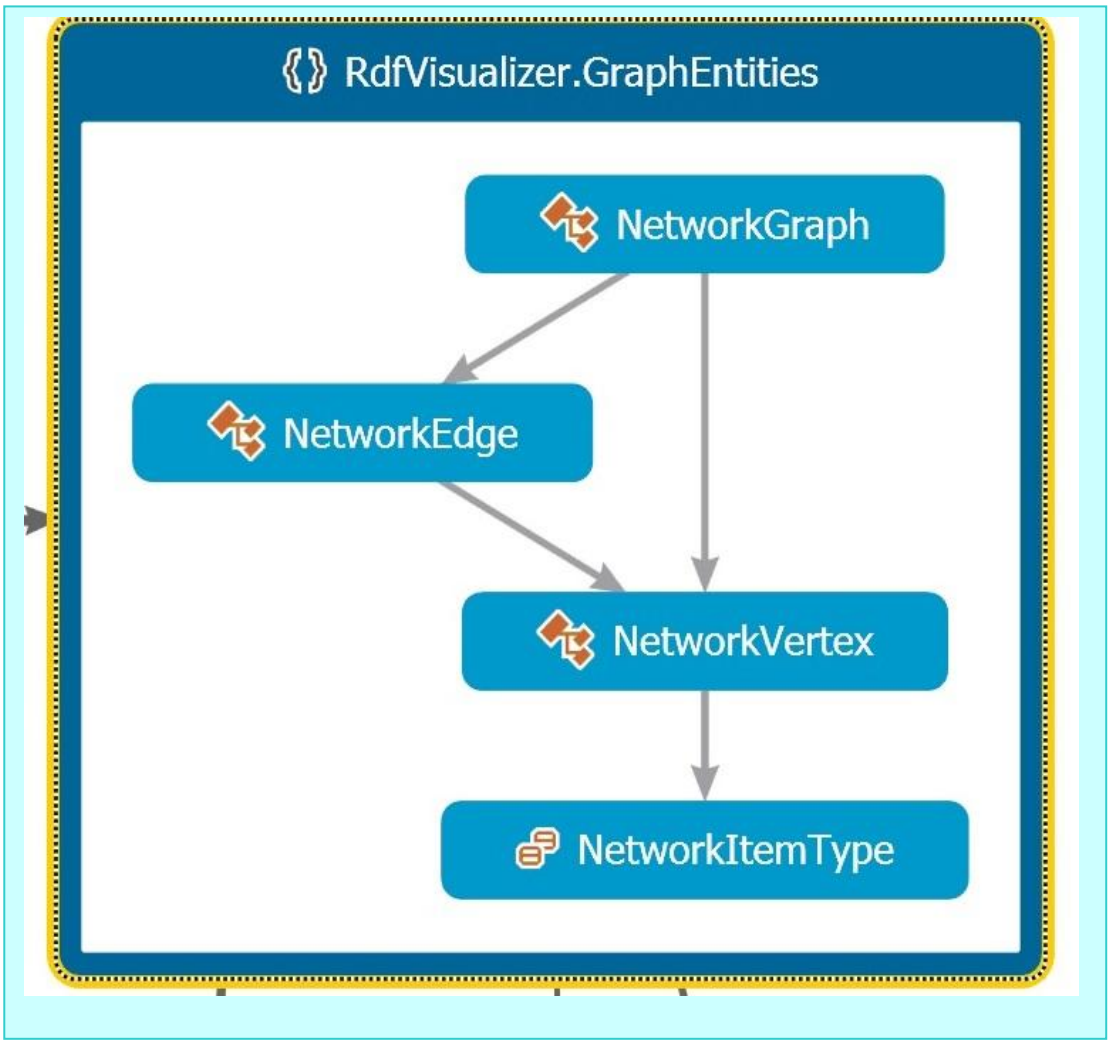


FIGURE 7.3: MODELS OF GRAPHENTITIES

NetworkEdge

This class serves as an application model. It describes all the necessary properties, methods and events which should be implemented for construction and visualization of edges between vertices and for search of the optimal way between vertices of the graph**.** Part of the NetworkEdgel class can be seen in listing 7.1.

```
public class NetworkEdge : TaggedEdge<NetworkVertex,string>,
INotifyPropertyChanged
    {
        private bool _isBestWay;
        private string _description;
        private bool _isSelected;
        public string Description
        {
            get { return _description; }
            set
            {
                _description = value;
                OnPropertyChanged("Description");
            }
        }
        public bool IsBestWay
        {
            get { return _isBestWay; }
            set
            {
                _isBestWay = value;
                OnPropertyChanged("IsBestWay");
            }
        }
        public bool IsSelected
        {
            get { return _isSelected; }
            set
            {
                _isSelected = value;
                OnPropertyChanged("IsSelected");
            }
        }
        private ICommand _edgeChooseCommand;
        public ICommand EdgeChooseCommand
        {
            get { return _edgeChooseCommand; }
            set { _edgeChooseCommand = value; }
        }
        public event EventHandler Selected;
        public event PropertyChangedEventHandler PropertyChanged;

        public NetworkEdge(NetworkVertex source, NetworkVertex target, string
description)
            : base(source, target, description)
        {
            Description = description;
            EdgeChooseCommand = new DelegateCommand(EdgeChoosedAction);
        }
        private void EdgeChoosedAction()
        {
            OnSelected();
        }
        protected virtual void OnSelected()
        {
            EventHandler handler = Selected;
            if (handler != null) handler(this, EventArgs.Empty);
        }
            .....
```

LISTING 7.1: NETWORK EDGE

NetworkVertex

This class represents a dynamic data collection which describes vertices of the graph. We will apply this description for creation of internal representation, visualization and editing of NDL documents. Part of the NetworkVertex class can be seen in listing 7.2.

```
public class NetworkVertex : INotifyPropertyChanged
{
    #region Private Fields
    private bool _isSearched;
    private IList<string> _liskTo;
    private ObservableCollection<NetworkVertex> _linkToVertexies;
    private string _portConnectedTo;
    private string _interfaceToDomain;
    private bool _isSelected;
    private string _label;
    private string _locatedAt;
    #endregion
    #region Public Properties
    public string LocatedAt
    {
        get { return _locatedAt; }
        set
        {
            _locatedAt = value;
            OnPropertyChanged("LocatedAt");
        }
    }
    public string Label
    {
        get { return _label; }
        set
        {
            _label = value;
            OnPropertyChanged("Label");
            OnPropertyChanged("About");
        }
    }
    public string About
    {
        get { return "#" + _label; }
    }
    public NetworkItemType NetworkItemType { get; private set; }
    public ObservableCollection<NetworkVertex> LinkToVertexies
    {
        get
        {
            return _linkToVertexies;
        }
        set
        {
            _linkToVertexies = value;
            OnPropertyChanged("LinkToVertexies");
        }
    }
      ......
```

LISTING 7.2: NETWORKVERTEX

7.5 The ViewModel class

Adding attribute to the property, the dependence of the container decides and implements the specified type after creating the View. Introduction of the **ViewModel** is installed directly in the contextual data (datacontext) **View**. RdfVisualizer contains the following ViewModels:

- DeviceDetailsViewModel - ViewModel for manipulation with object model of the graph
- MainWindowViewModel – ViewModel for a Window/UserControl that hosts the RdfVisualizer graphing controls. Listing 7.3

Using **PocGraphLayou**t we can create a custom LayoutTyppe for the custom graph. Listing 7.4

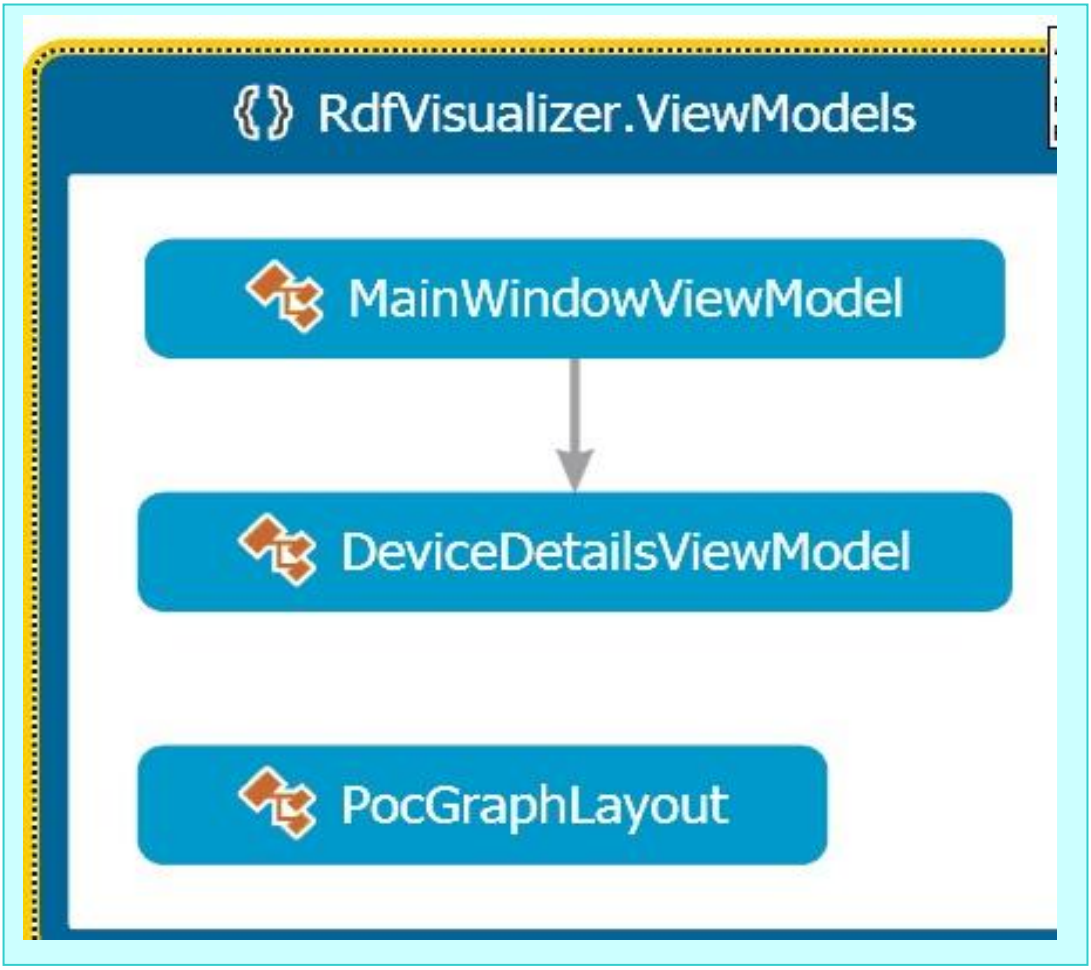


FIGURE 7.4: THE VIEWMODEL

```
namespace RdfVisualizer.ViewModels
{
    public class PocGraphLayout : GraphLayout<NetworkVertex, NetworkEdge,
NetworkGraph> { }
    public class MainWindowViewModel : INotifyPropertyChanged
    {
        #region Private Fields
        private NetworkGraph _graph;
        private readonly MainWindow _mainWindow;
        public DelegateCommand LoadRdfCommand { get; set; }
        private readonly ObservableCollection<NetworkVertex>
_networkVertices;
        private string _layoutAlgorithmType;
        private string _searchString;
        private NetworkVertex _vertexFrom;
        private NetworkVertex _vertexTo;
      // private NetworkEdge _selectedEdge;
        #endregion
        #region Public Properties
        public ICommand DeleteVertexCommand { get; set; }
        public DelegateCommand AddDeviceCommand { get; set; }
        public DelegateCommand AddEdgeCommand { get; set; }
                public string LayoutAlgorithmType
        {
            get { return _layoutAlgorithmType; }
            set
            {
                _layoutAlgorithmType = value;
                OnPropertyChanged("LayoutAlgorithmType");
            }
        }
        public NetworkGraph Graph
        {
            get { return _graph; }
            set
            {
                _graph = value;
                OnPropertyChanged("Graph");
            }
        }
        public ObservableCollection<NetworkVertex> NetworkVerticesFrom
        {
            get
            {
                return _networkVertices;
            }
        }
        public ObservableCollection<NetworkVertex> NetworkVerticesTo
        {
            get
            {
                return _networkVertices;
            }
        }
       .....
```

LISTING 7.3: MAINWINDOWVIEWMODEL

```
<Window x:Class="RdfVisualizer.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:controls="clr-
namespace:WPFExtensions.Controls;assembly=WPFExtensions"
        xmlns:controls1="clr-
namespace:GraphSharp.Controls;assembly=GraphSharp.Controls"
        xmlns:graphEntities="clr-namespace:RdfVisualizer.GraphEntities"
        xmlns:viewModels="clr-namespace:RdfVisualizer.ViewModels"
        xmlns:converters="clr-
namespace:GraphSharp.Converters;assembly=GraphSharp.Controls"
        xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
        Title="RDF Visualizer" Height="600" Width="800"
Loaded="OnMainWindowLoaded" WindowState="Maximized">
    <Window.Resources>
        <DataTemplate x:Key="DemoTemplate" DataType="{x:Type
graphEntities:NetworkVertex}">
            <StackPanel Orientation="Vertical" Margin="5" >
                <CheckBox IsChecked="{Binding IsSelected, Mode=TwoWay}" />
                <TextBlock  x:Name="title" Text="{Binding Path=About,
Mode=OneWay}" Foreground="White"/>
                <TextBlock FontSize="9" Text="{Binding Path=LocatedAt,
Mode=OneWay}" Foreground="White"/>
                <!--<TextBlock  Text="{Binding Path=Label, Mode=OneWay}"
Foreground="White"/>-->
                <StackPanel Orientation="Horizontal">
                    <Button Content="Edit" Command="{Binding
EditVertexCommand}" VerticalAlignment="Center" HorizontalAlignment="Left"/>
                    <Button Content="Delete" CommandParameter="{Binding}"
Command="{Binding DataContext.DeleteVertexCommand,
RelativeSource={RelativeSource Mode=FindAncestor,AncestorType={x:Type
controls:ZoomControl}}}" VerticalAlignment="Center"
HorizontalAlignment="Left"/>
                </StackPanel>
```

LISTING 7.4: COMMUNICATION OF DATA AND VIEW

Chapter 8

Base algorithms

8.1 Introduction

In this chapter we will talk about the basic algorithms, which we used for creation of this project. We will note the based libraries used for creation of internal representation and visualization of NDL documents.
We uses an open source library called GraphSharp [28] to layout the elements in canvas. This library provides several different layout algorithms for graphs. It contains some layout algorithms and a GraphLayout control for WPF applications.

8.2 Parsing NDL documents

To create the Efficient internal representation of the NDL document, we use The XML Document Object Model (DOM) [29]. The DOM presents an XML document as a tree-structure. Event-based XML parser allows to access the relevant data. Consider the example shown in listing 8.1.
We use the classes for LINQ to XML [30]. Also we use the System.Xml.XPath namespace contains the classes that define a cursor model for navigating XML information items [31]

```
OnPropertyChanged("NetworkVerticesFrom");
                OnPropertyChanged("NetworkVerticesTo");
                OnPropertyChanged("Graph");
                OnPropertyChanged("NetworkVerticesFrom");
                OnPropertyChanged("NetworkVerticesTo");
                var xml = File.ReadAllText(ofd.FileName);
                XNamespace rdfNamespace = "http://www.w3.org/1999/02/22-
rdf-syntax-ns#";
                XNamespace ndlNamespace =
"http://www.science.uva.nl/research/sne/ndl#";
                XNamespace domainNamespace =
"http://www.science.uva.nl/research/sne/ndl/domain#";
                var xdoc = XDocument.Parse(xml);
                var domains = xdoc.Descendants(domainNamespace +
"NetworkDomain");
                var devices = xdoc.Descendants(ndlNamespace + "Device");
                var nameTable = new NameTable();
                var namespaceManager = new XmlNamespaceManager(nameTable);
                namespaceManager.AddNamespace("ndl",
"http://www.science.uva.nl/research/sne/ndl#");
                foreach (var device in devices)
                {
                    var deviceNetworkName = device.Attribute(rdfNamespace
+ "about").Value.Replace("#", string.Empty);
                    var interfaces =
                        xdoc.XPathSelectElements("//ndl:Interface",
namespaceManager)
                            .Where(a => a.Attribute(rdfNamespace +
"about").Value.Contains(deviceNetworkName))
                            .SelectMany(
                                intf =>
                                    intf.Elements(ndlNamespace + "linkTo")
                                        .Select(linkTo =>
linkTo.Attribute(rdfNamespace + "resource").Value)).ToList();

                    var interfaceToDomain =
                        xdoc.XPathSelectElements("//ndl:Interface",
namespaceManager)
                            .Where(a => a.Attribute(rdfNamespace +
"about").Value.Contains(deviceNetworkName))
                            .Select(a =>a.Attribute(rdfNamespace +
"about").Value)
                            .FirstOrDefault();
                    if (interfaceToDomain != null)
                    {
                        var split = interfaceToDomain.Split(new[] { ':' },
StringSplitOptions.RemoveEmptyEntries);
                        interfaceToDomain = split.Length > 1 ? split[1] :
null;
                    }
                    _networkVertices.Add(new
NetworkVertex(deviceNetworkName, NetworkItemType.Device, interfaces,
interfaceToDomain));
                }
                foreach (var domain in domains)
                {
                    var deviceNetworkName = domain.Attribute(rdfNamespace
+ "about").Value.Replace("#", string.Empty);
                    var interfaces =
                        domain.Elements(domainNamespace + "hasDevice")
                            .Select(a => a.Attribute(rdfNamespace +
"resource").Value).ToList();
                    _networkVertices.Add(new
NetworkVertex(deviceNetworkName, NetworkItemType.Domain, interfaces)); }
```

LISTING 8.1: PARSING. NDL FILES(TOPOLOGY SCHEMA) WITH DOM

8.3 Manipulating NDL documents

For manipulations with the Graph we added additional ViewModel (DeviceDetailsViewModel). We have created an additional class, which describe the methods and objects to create new nodes of the graph, editing of existing nodes and saving the modified data in the NDL document[1]. Listing: 8.2

```
        public event PropertyChangedEventHandler PropertyChanged;

        public DeviceDetailsViewModel(DeviceDetails view, NetworkVertex
editedNetworkVertex)
        {
            EditedNetworkVertex = editedNetworkVertex;
            Label = editedNetworkVertex.Label;
            LocatedAt = editedNetworkVertex.LocatedAt;
            View = view;
            View.DataContext = this;
            SaveCommand = new DelegateCommand(Save);
            CloseCommand = new DelegateCommand(Close);
        }

        private void Close()
        {
            View.DialogResult = false;
            View.Close();
        }

        private void Save()
        {
            View.DialogResult = true;
            View.Close();
        }

        [NotifyPropertyChangedInvocator]
        private void OnPropertyChanged(string propertyName)
        {
            PropertyChangedEventHandler handler = PropertyChanged;
            if (handler != null) handler(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
```

LISTING 8.2: DEVICEDETAILSVIEWMODEL FOR MANIPULATIONS WITH THE GRAPH

[1] At creation of the editor, we consulted with a senior software developer.

8.4 Modification of the Dijkstra Algorithm

In Chapter 5, we considered the basic algorithms of search in the graph, we described their features and showed simple examples. For implementation of the Dijkstra algorithm we used QuickGraph. It provides generic directed/undirected graph data structures and algorithms for .NET.
QuickGraph comes with algorithms such as depth first search, breath first search, shortest path, k-shortest path, maximum flow, minimum spanning tree, least common ancestors, etc... [31]
The AlgorithmExtensions class contains several helper methods to execute the algorithm. We show how to solve this problem with QuickGraph in listing 8.3.

```
public void ShowBestWay()
        {
            if (VertexFrom != null && VertexTo != null)
            {
                IEnumerable<NetworkEdge> result;
                Graph.ShortestPathsDijkstra(EdgeWeights,
VertexFrom).Invoke(VertexTo, out result);
                if (result != null)
                {
                    foreach (var networkEdge in Graph.Edges)
                    {
                        networkEdge.IsBestWay = false;
                    }

                    foreach (var networkEdge in result)
                    {
                        networkEdge.IsBestWay = true;
                        var edge =
                            Graph.Edges.FirstOrDefault(
                                a => a.Source == networkEdge.Target &&
a.Target == networkEdge.Source);
                        if (edge != null)
                        {
                            edge.IsBestWay = true;
                        }
                    }
                }
            }
        }
```

LISTING 8.3: IMPLEMENTATION OF THE DIJKSTRA ALGORITHM

We developed the algorithm which modifies the classical Dijkstra's algorithm and adds parameters and limits which are important in our opinion.
Working principle of algorithm:

- calculates capacity for the interface;
- receives encoding type for the device;

- If encodings are not compatible, the algorithm will have to look for other ways, spending extra time.
- further, we can use standard methods of algorithm (dijkstra's algorithm).

```
private double EdgeWeights(NetworkEdge networkEdge)
        {
            double capacity = 1;
            if (networkEdge.Capacity != null &&
!string.IsNullOrEmpty(networkEdge.Capacity.CapacityValue))
            {
                double.TryParse(networkEdge.Capacity.CapacityValue, out
capacity);
            }

            string encodingTypeTarget = string.Empty;
            string encodingTypeSource = string.Empty;
            if (networkEdge.Source.Interface != null &&
networkEdge.Source.Interface.EncodingType != null)
            {
                encodingTypeSource =
networkEdge.Source.Interface.EncodingType.EncodingTypeValue;
            }
            if (networkEdge.Target.Interface != null &&
networkEdge.Target.Interface.EncodingType != null)
            {
                encodingTypeTarget =
networkEdge.Target.Interface.EncodingType.EncodingTypeValue;
            }

            return networkEdge.Target.NetworkItemType ==
NetworkItemType.Domain ||
    networkEdge.Source.NetworkItemType == NetworkItemType.Domain
                ? 100000000000
    : capacity*(encodingTypeSource.Equals(encodingTypeTarget) ? 1.0 : 0.5);
        }
```

LISTING 8.4: IMPLEMENTATION OF THE DIJKSTRA ALGORITHM (MODIFICATION)

Conclusions

The aim of this thesis was to create the Tools for NDL Elaboration. Created application project offers tools to create and edit generic topology schemes NDL documents.
These documents are shown as visualized trees. This visualization can be modified. The visualized documents can be exported to various formats including TXT, XML and RDF.
NDL topology schemas can also be edited using this application. Using internal representation of NDL elements, the user does not need to remember the names of the elements available. RdfVisualizer is capable of opening and editing huge NDL files. RdfVisualizer is one of the first attempts in creating tools to work with NLD documents. But I think that, RdfVisualizer cannot to compete with professional solutions which are on the market nowadays. Those solutions are being developed by the whole teams of professional developers.
We tried to show how NDL editing can be done in a different way and can be used for example for creating visualizations of simple NDL documents, editing internal representation of NDL and finding the shortest path in NDL documents.
We are sure that there are many ways how this project could be enhanced. The following list shows ideas which could make this project a better application:

- Add simultaneous text and visual editing of other types of NDL documents,
  first of all, it is NDL layer schema and NDL domain schema.

- Improve user-friendliness of elements editing.

- Allow users to better customize the appearance of visualized elements.

- Optimize performance of the application when there are many (thousands) of elements on canvas.

- Add more algorithms of search (the Floyd–Warshall algorithm, the Bellman-Ford algorithm, A*, ...).

- Improve of internal representation of NDL elements.

Bibliography

[1]. Global Lambda Integrated Facility (GLIF): http://www.glif.is/.

[2]. Cees de Laat, Erik Radius, and Steven Wallace: The Rationale of the Current Optical Networking Initiatives. Future Generation Computer Systems, 19(6):999–1008 (August 2003). http://www.sciencedirect.com/science/article/

[3]. HP Openview. http://openview.hp.com/ .

[4]. Dynamic Resource Allocation Controller (DRAC). http://www.nortel.com/drac/

[5]. Iljitsch van Beijnum: BGP. O'Reilly Media, Inc. (2002). ISBN 9780596002541 .

[6]. X. Cao, V. Anand, and C. Qiao. Waveband Switching in Optical Networks. In IEEE Communications Magazine Apr. 2003. doi:10.1109/MCOM.2003.1193983 .

[7]. Characteristics of synchronous digital hierarchy (SDH) equipment functional blocks. Recommendation ITU-T G.783, International Telecommunication Union (ITU), Feb. 2004. http://www.itu.int/rec/T-REC-G.783/ .

[8]. Synchronous Optical Network (SONET) - Basic Description including Multiplex Structure, Rates, and Formats. Standard T1.105, American National Standards Institute (ANSI),2001. http://webstore.ansi.org/RecordDetail.aspx?sku=T1.105-2001 .

[9]. Interfaces for the Optical Transport Network (OTN). Recommendation ITU-T G.709 / ITU-T Y.1331, International Telecommunication Union (ITU), Mar. 2003. http://www.itu.int/rec/T-REC-G.709/ .

[10]. Architecture of Optical Transport Networks. Recommendation ITU-T G.872, international Telecommunication Union (ITU), Nov. 2001.http://www.itu.int/rec/T-REC-G.872/ .

[11]. B-ISDN asynchronous transfer mode functional characteristics. Recommendation ITU-T I.150, International Telecommunication Union (ITU), Feb. 1999. http://www.itu.int/rec/T-REC-I.150/ .

[12]. Franco Travostino: Using the Semantic Web to Automate the Operation of a Hybrid Internetwork. In GridNets conference proceedings (October 2005).

[13]. The Semantic Web. http://www.w3.org/2001/sw/ .

[14]. Resource Description Framework (RDF). http://www.w3.org/RDF/ .

[15]. Eric Prud'hommeaux and Andy Seaborne: SPARQL Query Language for RDF (2005). http://www.w3.org/TR/rdf-sparql-query/ .

[16]. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (June 1999). Updated by RFC 2817, http://www.ietf.org/rfc/rfc2616.txt .

[17]. T. Berners-Lee, R. Fielding, and L. Masinter: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard) (January 2005). http://www.ietf.org/rfc/rfc3986.txt.

[18]. Friend of a Friend (FOAF) Project. http://www.foaf-project.org/ .

[19]. Dublin Core Metadata Initiative. http://www.dublincore.org/.

[20]. D. Beckett. RDF/XML Syntax Specification. Recommendation, W3C, Feb. 2004. http://www.w3.org/TR/rdf-syntax-grammar/ .

[21]. The shortest-paths problém. http://urban-sanjoo.narod.ru/shortest_way.html

[22]. The Bellman–Ford algorithm. http://urban-sanjoo.narod.ru/bellman-ford.html

[23]. The algorithm of Dijkstra's. http://habrahabr.ru/post/111361/

[24]. The Floyd–Warshall algorithm. http://www.graphmagics.com/articles/all_shortest_paths.php

[25]. The Floyd–Warshall algorithm. Example. http://urban-sanjoo.narod.ru/floyd.html

[26]. WPF: MVVM.http://www.codeproject.com/Articles/36545/WPF-MVVM-Model-View-View-Model-Simplified

[27]. MVVM(Model-View-ViewModel): http://www.csharpcorner.com/uploadfile/nipuntomar/mvvm-in-wpf/

[28]. Graphsharp: http://graphsharp.codeplex.com/

[29]. XML DocumentObjectModel(DOM):http://msdn.microsoft.com/en-us/library/hf9hbf87%28v=vs.110%29.aspx

[30]. System.Xml.Linq: http://msdn.microsoft.com/enus/library/system.xml.linq%28v=vs.110%29.aspx

[31]. QuickGraph, Graph Data Structures And Algorithms for .NET : http://quickgraph.codeplex.com/

[32]. A computer network : http://en.wikipedia.org/wiki/Computer_network

[33]. XML Data Visualization, Roman Betík: http://www.ksi.mff.cuni.cz/~holubova/bp/Betik.pdf

[34]. Network management: http://en.wikipedia.org/wiki/Network_management

[35]. A Semantic Model for Complex Computer Networks Jeroen van der Ham PhD Thesis : http://staff.science.uva.nl/~vdham/research/publications/vdham-phdthesis

[36]. Framework for Path Finding in Multi-Layer Transport Networks, Freek Dijkstra PhD Thesis:http://www.macfreek.nl/work/Dijkstra-multilayer-pathfinding.pdf

[37]. Path selection in multi-layer networks Fernando Kuipers , Freek Dijkstra: http://staff.science.uva.nl/~fdijkstr/publications/multilayer-pathselection.pdf

List of Figures

List of Listings

List of Abbreviations

OWL	Web Ontology Language
RDF	Resource Description Framework
NDL	Network Description Language
XML	Extensible Markup Language
W3C	World Wide Web Consortium
LINQ	Language Integrated Query
DOM	Document Object Model
LAN	Local Area Network
WAN	Wide Area Network
MAN	Metropolitan area network
SONET	Synchronous Optical Networking
WLAN	Wireless Local Area Network
SDH	Synchronous Digital Hierarchy
TDM	Time Division Multiplexing
WDM	Wavelength Division Multiplexing

Appendix A

Content of the attached CD

/SourceCode/	Contains the complete source code in the form of solution in Visual Studio 2012
/Documentation/	Contains HTML documentation generated from comments in the source code
/Release/	Contains the installer of project with all required dependencies
/Thesis/	Text of this thesis in PDF format