Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Burim Mulici

Topologiebasierte Visualisierung von Netzwerkmetriken der System-Level Simulation

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Burim Mulici

Topologiebasierte Visualisierung von Netzwerkmetriken der System-Level Simulation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Franz Korf
Zweitgutachter: Prof. Dr. Wolfgang Fohl

Eingereicht am: 3. Mai 2016

Burim Mulici

Thema der Arbeit
Topologiebasierte Visualisierung von Netzwerkmetriken der System-Level Simulation

Stichworte
Echtzeit-Ethernet-Netzwerk, Simulation, OMNeT++, Scalar, Vector, Eclipse Plug-In, Metrik

Kurzzusammenfassung
Die vorliegende Bachelorarbeit befasst sich mit der Visualisierung von Netzwerktopologien und Metriken von Echtzeit-Ethernet-Netzwerken. Für die Darstellung der Topologien und Metriken wird der Eclipse Plug-In Topology Analyzer entwickelt, welcher sich in die OMNeT++ Simulationsumgebung integrieren lässt. Das Plug-In muss in der Lage sein, OMNeT++ Netzwerke abzubilden. Um dies zu ermöglichen, muss zunächst die Methode zur Erfassung der Informationen bezüglich der Topologie und der Simulationsergebnisse entschieden werden. Am Ende werden die Ergebnisse der Evaluierung zum Plug-in aufgeführt.

Burim Mulici

Title of the paper
Topology based visualization of network metrics for System-Level simulation

Keywords
Real-Time network, Simulation, OMNeT++, Scalar, Vector, Eclipse Plug-In, Metric

Abstract
The bachelor thesis deals with the visualization of network topologies and metrics of real-time Ethernet networks. In order to illustrate the topologies and their metrics the Eclipse plug-in Topology Analyzer is developed which can be integrated into the OMNeT ++ simulation environment. The plug-in must be able to reflect OMNeT++ networks. To make this possible, first the method for data collecting has to be decided. At the end the results of the evaluation of the plug-in are listed.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1. Einleitung

In der heutigen Zeit bekommen Echtzeitnetzwerke immer mehr Aufmerksamkeit. Diese werden in vielen Bereichen eingesetzt. Dazu gehört die Kommunikation im Auto, im Flugzeug oder bei industriellen Automatisierungsanwendungen. Die vorliegende Arbeit ist in der Communication over Realtime Ethernet (CoRE) Arbeitsgruppe [CoRE Research Group, a] an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg entstanden. Diese befasst sich mit der Realisierung von Echtzeit-Ethernet-Kommunikation im Auto und Flugzeug.

Auch in Zukunft ist Software die Technik, um komplexe Algorithmen im Kraftfahrzeugen und Flugzeugen umzusetzen. Der Software-Umfang ist nahezu exponentiell wachsend und wird durch die Funktionszunahme in Richtung Sicherheit und Zuverlässigkeit getrieben. Der Elektronik- und Softwareanteil ist ein wesentlicher Innovationsmeter, d.h. etwa 90% der Innovationen im Fahrzeug sind von der Elektronik geprägt. Somit werden Fahrzeuge und Flugzeuge stetig komplexer. [Schäuffele und Zurawka, 2006].

Echtzeitnetzwerke setzen insbesondere ein klar definiertes Zeitverhalten, welches unter allen Betriebsbedingungen gewährleistet ist, voraus. Um das Verständnis von solchen komplexen Systemen zu erleichtern, werden verschiedene Methoden angeboten. Eine Methode ist die Visualisierung solcher komplexer Netzwerken. Durch die Visualisierung können intuitive Erkenntnisse vermittelt werden, welche zur Analyse und Kommunikation der dargestellten Daten dienen. Daher ist die Visualisierung von Netzwerktopologien und ihrer Metriken ein relevantes Hilfsmittel. Dies ermöglicht die Überprüfung des Verhaltens des Netzwerkes, um mögliche Fehler im Netzwerk aufdecken zu können.

Ziel der Arbeit

Das Ziel dieser Arbeit setzt sich aus zwei Teilen zusammen. Zunächst müssen die relevanten Informationen bezüglich der Topologie und die Simulationsergebnisse aus der OMNeT++ Umgebung aufgezeichnet werden. Dementsprechend muss der Plug-In Topology Analyzer programmiert werden, der die Topologie Informationen und Simulationsergebnisse einliest und visualisiert. Für die Darstellung der Topologie und ihrer Metriken müssen Semantiken entworfen werden, welche eine exakte Analyse der OMNeT++ Netzwerke ermöglicht und im besten Fall auf Fehlerquellen hinweist.

Struktur der Arbeit

In Kapitel 2 werden die Grundlagen erläutert, die zu einem besseren Verständnis der nachfolgenden Kapitel dienen. In diesen werden die Simulationsumgebung OMNeT++, die Metriken eines Netzwerkes und die Eclipse Plug-Ins erläutert. Anschließend werden alle Anforderung an das Programm, die Datenerfassung und die Visualisierung festgelegt, die für die Erstellung der Konzepts notwendig sind. Im Anschluss folgt das Konzept. In Kapitel 5 wird die Implementierung des Konzepts erläutert. Daraufhin wird der Eclipse Plug-In Topology Analyzer evaluiert und zum Schluss wird die gesamte Arbeit zusammengefasst.

2. Grundlagen

In diesem Kapitel werden Grundlagen vermittelt, die zum besseren Verständnis dieser Arbeit dienen. Zuerst werden die Grundlagen zur OMNeT++ Simulationsumgebung dargestellt. Dann werden ausgewählte Netzwerkmetriken erläutert. Zuletzt werden die Eclipse-Umgebung und deren Plug-Ins vorgestellt.

2.1. Simulationsumgebung OMNeT++

Objective Modular Network Testbed (OMNeT++) ist eine Entwicklungsumgebung für Netzwerksimulationen [OMNeT++ Community, a]. OMNeT++ arbeitet eventbasiert und wird hauptsächlich zur Simulation von Kommunikationsnetzwerken eingesetzt. Sie beinhaltet eine modulare, eine erweiterbare und eine komponentenbasierte C++ Bibliothek. Mithilfe dieser Bibliothek können beliebige Module mit unterschiedlichen Verhalten umgesetzt werden. Dies dient dazu, ein auf eigene Anforderungen zugeschnittenes Netzwerk zu erstellen und zu simulieren. OMNeT++ basiert auf Ecplise und kann unter Linux, Windows und Mac OS X betrieben werden [Varga und Hornig, 2008].

2.1.1. Aufbau einer OMNeT++ Simulation

Eine OMNeT++ Simulation besteht aus mehreren Modulen, die über Gates miteinander kommunizieren. Jedes Gate besitzt einen Input- und einen Output-Channel. Mithilfe dieser Channels können Module untereinander Nachrichten austauschen. In Abbildung 2.1 sind zwei dieser Module zu sehen. Die Module sind über die Input-/Output-Channels ihrer Gates miteinander verbunden [vgl. Manual OMNeT++ Community, b].

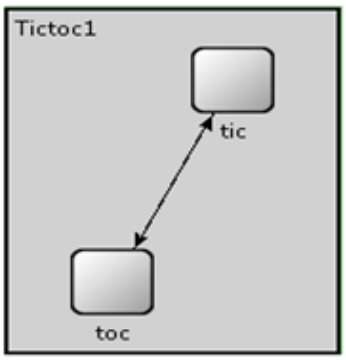


Abbildung 2.1.: Verbindung zweier Module [OMNeT++ TicToc Example]

In der OMNeT++ Umgebung gibt zwei verschiedene Typen von Modulen:

- Simple Module, die in anderen Modulen enthalten sind wie z.B. Tic bzw. Toc aus Abbildung 2.1.
- Compound Module, die sich aus eine Menge von Modulen und deren Simple Modulen zusammensetzen, wie z.B das TicToc1 Modul aus Abbildung 2.1.

Die Module werden in der Network Description (NED) Sprache beschrieben. In Listing 2.1 wird zuerst das Simple Module von tic bzw. toc definiert. Danach wird das Netzwerk TicToc1 definiert, das die beiden Simple Module tic und toc enthält.

```
simple Txc1{            //Deklaration eines Modules
    gates:
        input in;
        output out;
}
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
network Tictoc1{
    submodules:
        tic: Txc1;
        toc: Txc1;
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

Listing 2.1: NED-Code TicToc Beispiel [OMNeT++ TicToc Example]

2.2. Metriken

Der Begriff Metrik kommt aus dem Griechischen und steht für „Messen“. Im IT-Bereich werden Metriken als standardisierte Messgrößen verwendet. Die Effizienz eines Netzwerkes wird anhand ihrer Quality of Service (QoS) Parameter bestimmt. Die typischen QoS-Parameter sind die folgenden: Latenz, Jitter und Paketverlustrate. Außerdem sind die Metriken Buffergröße, Linksauslastung, Bandbreite und Kapazität für ein Netzwerk von Bedeutung und können Einfluss auf die QoS-Parameter haben [Evans und Filsfils, 2007].

2.2.1. Typen der Metriken

Im automobilen Netzwerk gibt es verschiedene Metriken, die für die Bewertung der Effizienz des Netzwerkes von Bedeutung sind. In diesem Abschnitt werden diese Metriken vorgestellt und kurz erläutern.

Kapazität
Die Kapazität beschreibt die maximal mögliche Bandbreite auf einem Link [vgl. Fabian Kempf, 2008, Seite 2].

Metrik:	Kapazität
Einheit:	Mb/s
Größenordnung:	54,7Mb/s - 97,5Mb/s bei einem Ethernet-Link mit 100Mb/s

Tabelle 2.1.: Kapazität

Linkauslastung

Diese Metrik beschreibt die Auslastung auf einem einzelnen Link im Verhältnis zur maximalen Bandbreite und kann prozentual angegeben werden. Eine absolute Angabe in Mb/s ist ebenfalls möglich. Die Linkauslastung geht über den gesamten Zeitraum der Simulation [vgl. Fabian Kempf, 2008, Seite 3].

Metrik:	Linkauslastung
Einheit:	Prozent (%) , Mb/s
Größenordnung:	Von 0% -100% , 0Mb/s – Maximale Bandbreite

Tabelle 2.2.: Linkauslastung

Verfügbare Bandbreite

Diese Metrik gibt die noch verfügbare Bandbreite auf einem einzelnen Link an. Die verfügbare Bandbreite ist das Gegenstück zur Linkauslastung. Dabei kann die Darstellung dieser Metrik prozentual oder in Mb/s angegeben werden [vgl. Fabian Kempf, 2008, Seite 4].

Metrik:	Verfügbare Bandbreite
Einheit:	Prozent (%) , Mb/s
Größenordnung:	Von 100% - 0% , 100Mb/s – 0Mb/s

Tabelle 2.3.: Verfügbare Bandbreite

Latenz

Die Latenz beschreibt die Verzögerung einer Botschaft vom Absenden bis zum Empfangen. Die Latenz hat großen Einfluss auf die Quality of Service (QoS) in einem Netzwerk. Entscheidend für diese Metrik sind die minimale, die maximale und die durchschnittliche Latenz. Die minimale Latenz gibt die kleinste Verzögerung einer Nachricht zwischen zwei Punkten im Netzwerk an. Die maximale Latenz ist das Gegenstück und gibt die längste Verzögerung einer Nachricht an. Die durchschnittliche Latenz gibt die durchschnittliche Verzögerung aller Pakete in einem Zeitintervall an. In dieser Metrik ist das Zeitintervall die gesamte Simulationszeit [vgl. Fabian Kempf, 2008, Seite 5].

Metrik:	Latenz
Einheit:	Sekunde (s)
Größenordnung:	Mikrosekunden (μs)

Tabelle 2.4.: Latenz

Jitter

Der Jitter ist die Variabilität der Latenz und wird von der Latenz abgeleitet. Der Jitter gehört ebenfalls zu den QoS-Parametern [vgl. Fabian Kempf, 2008, Seite 6].

Metrik:	Jitter
Einheit:	Sekunde (s)
Größenordnung:	Mikrosekunden (μs)

Tabelle 2.5.: Jitter

Paketverlustrate im Buffer

Die Metrik Paketverlustrate im Buffer gibt das Verhältnis der Pakete an, die durch einen Bufferüberlauf verloren gehen. Dabei werden alte Pakete mit neu ankommenden Paketen überschrieben. Diese Pakete werden als „Dropped“bezeichnet. Der Verlust von Paketen führt zum erneuten Senden und damit wiederum zu einem langsamen Netzwerk [vgl. Fabian Kempf, 2008, Seite 6].

Metrik:	Paketverlustrate im Buffer
Einheit:	Prozent (%)
Größenordnung:	0% - 100%

Tabelle 2.6.: Paketverlustrate im Buffer

Buffergröße

Diese Metrik beschreibt die Anzahl an zwischengespeicherten Frames, die sich in einem Switch oder Endgerät befinden. Ist der Buffer überfüllt, steigt die Latenz der neu ankommenden Nachrichten. Wenn dies der Fall ist, ist das Netzwerk für hohe Last nicht geeignet [vgl. Fabian Kempf, 2008, Seite 7].

Metrik:	Buffergröße
Einheit:	Anzahl Frames (%)
Größenordnung:	1 - 50 Frames

Tabelle 2.7.: Buffergröße

2.3. Eclipse-Umgebung

In diesem Abschnitt wird zunächst die Programmierumgebung Eclipse erläutert, um eine bessere Übersicht über Plug-Ins zu bekommen. Dann wird auf die Entwicklung eines Plug-Ins und die Herangehensweise eingegangen.

2.3.1. Eclipse

Eclipse ist eine Java-basierte Open Source-Plattform, die ursprünglich nur als Integrated Development Environment (IDE) entwickelt wurde. Aktuell ist Eclipse nicht nur eine IDE, sondern auch eine Tool-Integrationsplattform, die je nach Aufgabenstellung erweitert werden kann. Die Erweiterungen werden mit Plug-Ins vorgenommen. Dies ermöglicht dem Nutzer eine Vielzahl von Tools in einer Benutzeroberfläche mit dem gleichen Erscheinungsbild zu verwenden. Eine Übersicht über den Aufbau von Eclipse ist in Abbildung 2.2 zu sehen. Die Grundfunktionalitäten sind in der „Platform Runtime“ der Eclipse Plattform. Diese beinhaltet die Workbench, die Werkzeuge für die Entwicklung von graphischen Benutzeroberflächen (SWT und JFace), den Workspace, Hilfe und Team. Alle anderen Funktionalitäten werden vom Nutzer als Plug-In eingebunden, wie z.B. das Java Development Tooling (JDT), das Plug-In Developer Enviroment (PDE) und andere/eigene Plug-Ins [vgl. Daum, Berthold, 2008, S.11-13].

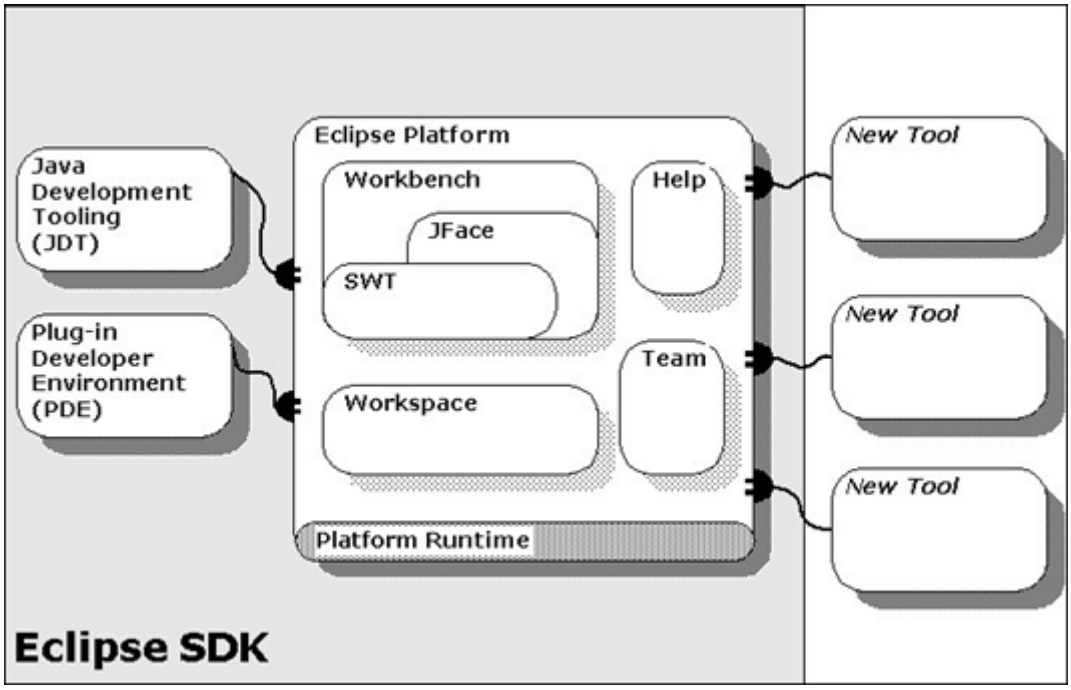


Abbildung 2.2.: Eclipse-Plattform [Eclipse Documentation]

2.3.2. Eclipse-Plug-In

Ein Plug-In ist die Erweiterung der Eclipse-IDE. Bei der Entwicklung eines eigenen Plug-Ins wird der geschriebene Code in eine modulare, erweiterbare und teilbare Einheit verpackt [Eclipse Foundation]. Für die Entwicklung eines Plug-Ins wird die Plug-In Developer Environment benötigt und diese erfolgt dann über die Extensions und Extensionpoints.

Extension: Das Plug-In bietet eine Erweiterung (extension) an, basierend auf dem Erweiterungspunkt. Die Erweiterungen können Codes oder Daten sein [vgl. Vogel und Milinkovich, 2015, S. 633].

Erweiterungspunkt: Das Plug-In stellt eine API bereit und definiert den Erweiterungspunkt (extensionpoint). Dies ermöglicht anderen Plug-Ins, sich mit ihren Extensions

mit dem Erweiterungspunkt zu verbinden und ihre Funktionen zu nutzen. Die Erweiterungspunkte müssen die Extension bewerten können. Dazu beinhalten sie den dafür verfügbaren Code, um dies zu ermöglichen [vgl. Vogel und Milinkovich, 2015, S. 633].

Dependencies: Die Dependencies beinhalten Plug-Ins, die am eigenen Plug-In-Code mit Funktionalitäten beitragen und für das Kompilieren erforderlich sind [vgl. Daum, 2007, S. 34].

Des Weiteren besitzt das Plug-In ein plugin.xml File mit den Informationen zu den Extensions und Extensionpoints. Diese wird in der Extensible Markup Language (XML) Sprache beschrieben. Die Plug-In-Informationen werden in der MANIFEST.MF definiert und beinhalten z.B. Informationen zur Versionsnummer, zu den benötigten Bibliotheken etc. Das Standart Widget Toolkit (SWT) und die JFace-Werkzeuge stehen optional für die Programmierung der Benutzeroberfläche zu Verfügung. Die grundlegenden Schritte zur Implementierung eines Plug-Ins werden in fünf Schritten definiert [vgl. Shavor, Sherry u. a., 2004, Seite 221]:

1. Wo soll das Plug-In integriert werden?
2. Welche Anforderung sollen die Extensionpoints erfüllen?
3. Deklaration der Manifest-Datei.
4. Implementierung der Funktionalitäten für die Erweiterung.
5. Installation des Plug-Ins.

Weitere Informationen dazu in Kapitel 5 Implementation.

3. Anforderung

In diesem Kapitel werden die Anforderungen erläutert, die für das Konzept und die Implementierung dieser Arbeit wichtig sind. Das Programm Topology Analyzer soll entwickelt werden, das die Topologie und die dazugehörigen Metriken von Echtzeit-Ethernet-Netzwerken visuell darstellt. Im Mittelpunkt stehen die Anforderungen für die Erfassung der Netzwerkinformationen sowie die Anforderungen für die Visualisierung des Netzwerkes.

3.1. Erfassung der Topologie-Daten

Der Topology Analyzer soll ein existierendes OMNeT++ Netzwerk visuell darstellen. Für die Visualisierung des Netzwerkes ist die Erfassung der Topologie-Daten notwendig. Folgende Informationen müssen für die Visualisierung der Topologie beachtet werden:

- Typ der Module
- Name der Module
- Position der Module
- Verbindung der Module

Des Weiteren muss das Format, in dem die Netzwerk-Informationen enthalten sind, ausgewählt werden. Dabei muss beachtet werden, dass alle Netzwerkinformationen, die für die Visualisierung der Topologie wichtig sind, abgespeichert werden.

3.2. Erfassen der Metrik-Daten

Ein OMNeT++ Netzwerk erzeugt nach dem Ende der Simulation eine große Menge an Ergebnissen. Die Ergebnisse beinhalten eine Menge von Signalen und Werten, die gespeichert werden müssen. Der Umfang kann je nach Simulationsdauer immer größer werden, sodass eine gezielte Filterung der Daten notwendig wird. In Kapitel 2.2 wurden alle Metriken vorgestellt, die für die Visualisierung beachtet werden müssen. Für die Erfassung der Metrik-Daten müssen folgende Punkte beachtet werden:

- Welches Modul der Absender bzw. Empfänger ist.
- Um welchen Nachrichtentyp es sich handelt (avbA/B, TT, etc.)
- Die Metriktypen und Metrikwerte zu bestimmten Simulationszeitpunkten:
  - Die Latenz der Nachrichtentypen (avbA/B, TT, etc.), sowie der durchschnittliche, der minimale und der maximale Wert.
  - Der Jitter der Nachrichtentypen muss errechnet werden
  - Die Füllstände der Buffer-Queues der Module für die einzelnen Nachrichtentypen
  - Die Linkauslastung und Bandbreite zwischen physikalischen Verbindungen.

Eine weitere Anforderung ist das Filtern der Informationen, sodass nur die benötigten Metrikdaten aufgezeichnet werden, die für die Visualisierung wichtig sind.

3.3. Anforderungen an den Topology Analyzer

In diesem Abschnitt werden die Anforderungen an den Topology Analyzer erläutert. Zunächst werden die Anforderungen für die Entwicklung der GUI erläutert. Anschließend werden die Anforderungen für das Einlesen der Daten und zuletzt für die Darstellung der Topologie und ihrer Metriken beschrieben.

3.3.1. Anforderung an das Topology Analyzer Plug-In

Das zu entwickelnde Programm wird als Plug-In in die OMNeT++ Umgebung eingebunden. Diese muss folgende Anforderungen erfüllen:

- Die OMNeT++ Umgebung bietet dem Nutzer bereits verschiedene Formate für verschiedene Netzwerkinformationen an. Das Programm sollte in der Lage sein, OMNeT++ Netzwerk-Daten (Topologie und Metriken) über das ausgewählte OMNeT++ Format einlesen zu können (z.B .xml, .sca, .vec, .elog)
- Die eingelesenen Netzwerkdaten sollen in der entwickelten GUI visualisiert werden (z.B mit Hilfe von SWT, AWT oder Swing)
- Über das visualisierte Netzwerk muss der Nutzer die Metrik-Informationen gezielt auswählen und anzeigen können.
- Die bestimmte Darstellung von Metrik-Informationen sollen anhand eines Viewpoints an- und ausgeschaltet werden können.
- Das Programm soll als Eclipse Plug-In entwickelt werden, um dies in die OMNeT++ Umgebung einbinden zu können.
- Das Plug-In muss für den Nutzer einfach bedienbar in der OMNeT++ Umgebung eingebunden werden (z.B auf der Menüleiste).

3.3.2. Anforderung für die Datenerfassung im Programm

Wenn alle Netzwerk-Informationen erfasst und in den ausgewählten Formaten gespeichert sind, müssen für das Einlesen der Daten folgende Anforderungen erfüllt werden:

- Der Standort des Einlesens der Daten in die GUI muss für den Nutzer leicht verständlich sein. Der Nutzer muss in der Lage sein zu wissen, welche Daten erforderlich sind.
- Das Programm muss in der Lage sein, die Netzwerk-Informationen anhand eines ausgewählten Parsers auszulesen.
- Das Parsen der Informationen kann je nach Simulationsdauer und Komplexität des Netzwerkes länger dauern. Somit muss dem Nutzer mit einer Meldung signalisiert werden, dass die Informationen zu dem Zeitpunkt geparst werden.

3.3.3. Anforderung für die Darstellung

Nachdem der Nutzer die Topologie-Informationen eingelesen hat, muss dieser in der Lage sein, die Metriken auf der Topologie des Netzwerkes effizient zu beobachten. Hierfür gibt es für die Darstellung der Metriken verschiedene Anforderungen:

- Die Topologie des Netzwerkes muss beim Topology Analyzer exakt denselben Aufbau haben wie das OMNeT++ Netzwerk.
- Die Netzwerkmetriken müssen an die graphischen Objekte des visualisierten Netzwerkes gebunden werden.
- Anhand der Metriken eines Moduls werden Statistiken über die gesamte Simulationszeit erzeugt und abgebildet.
- Netzwerk-Informationen müssen mit einem Mausklick angezeigt werden können.

4. Konzept

In diesem Kapitel wird das Konzept erarbeitet, welches für die Implementierung des Topology Analyzer verwendet wurde. Zuerst werden die verschiedenen Methoden zur Datenerfassung von OMNeT++ Ergebnissen erläutert. Dementsprechend wird die Wahl für die geeignete Methode zur Datenerfassung getroffen. Zuletzt wird das Konzept des Topology Analyzer erläutert.

4.1. Konzept zur Datenerfassung

Dieser Abschnitt befasst sich zuerst mit der Erfassung der Topologie-Daten. Anschließend wird die Erfassung der Metrik-Daten beschrieben.

4.1.1. Topologie Daten

Die Topologie eines OMNeT++ Netzwerkes wird mit der NED-Sprache beschrieben (siehe Abschnitt 2.1.1). Als Beispiel wird das OMNeT++ Netzwerk in Abbildung 4.1 verwendet, dies besteht aus einem Switch und drei Nodes.

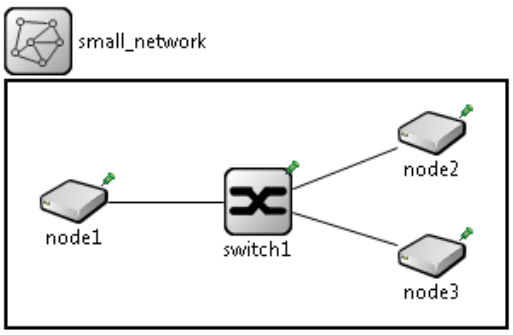


Abbildung 4.1.: small_network example aus dem CoRE4INET [CoRE Research Group, b]

Ein Ziel dieser Arbeit ist es, OMNeT++ Netzwerke zu visualisieren (graphisch darzustellen). Aus diesem Grund müssen die Topologie-Daten aus der OMNeT++ Simulation exportiert werden. Unter Topologie-Daten versteht man [vgl. Manual OMNeT++ Community, b, Kapitel 11]:

- Namen der Module
- Typen der Module
- Positionen der Module
- Farben der Module
- Verbindung der Module sowie Farben, Breiten und Linientypen der Verbindungen
- Breite und Länge der Module, etc.

```
1  package core4inet.examples.IEEE8021Q.small_network;
2  import inet.node.ethernet.Eth100M;
3  //Small sample network with three hosts and
4  //one switch that shows how to configure the IEEE 802.1Q Model.
5  //Configuration in omnetpp.ini - 80ns Tick length
6  network small_network{
7  parameters:
8  //Background color of the small_network example
9  @display("bgb=,,white");
10 double measure_interval @unit(s) = default(1s);
11 submodules:
12 node1: Node1 {
13 //position of the Module
14 @display("p=39,70");
15 }
16
17 node2: Node2 {
18 //position of the Module
19 @display("p=249,30");
20 }
21
22 node3: Node3 {
23 //position of the Module
24 @display("p=249,102");
25 }
```

```
26
27 switch1: Switch1 {
28 parameters:
29 //position of the Module
30 @display("p=147,70");
31 gates:
32 ethg[3];
33 }
34 connections:
35 // The Connection of the Modules. The example uses a simple star topology
36 node1.ethg <--> Eth100M {  length = 20m; } <--> switch1.ethg[0];
37 node2.ethg <--> Eth100M {  length = 10m; } <--> switch1.ethg[1];
38 node3.ethg <--> Eth100M {  length = 5m; } <--> switch1.ethg[2];
39 }
40 }
```

Listing 4.1: small_network.ned aus dem CoRE4INET [CoRE Research Group, b]

In Listing 4.1 wird der NED-Code für den Aufbau des Netzwerkes aus Abbildung 4.1 gezeigt. In den Zeilen 7 bis 33 werden die einzelnen Simple Module des Netzwerks mit ihren Positionen definiert. In den Zeilen 34 bis 39 werden die Verbindungen zwischen den Modulen definiert. Für die Erfassung der Topologie-Daten stehen zwei Methoden zur Auswahl, die im Folgenden erläutert werden.

OMNeT++ XML Datei

Die OMNeT++ Umgebung liefert eine bereits integrierte Methode für den Export der Topologie-Daten. Das Format entspricht einer XML-Datei, die Informationen beinhaltet, die für die Visualisierung eines OMNeT++ Netzwerkes benötigt werden.

Vorteile

- Bereits integriert in der OMNeT++ Umgebung
- Standardisierte Methode in OMNeT++, bleibt erhalten und wird gepflegt, auch nach einer Versionsänderung der OMNeT++ Umgebung.
- Dokumentation über das XML-Format vorhanden [OMNeT++ Community, c]

Nachteile

- Beinhaltet Informationen, die nicht für die Visualisierung eines OMNeT++ Netzwerkes relevant sind.

Eigene XML-Datei

Eine weitere Methode ist die Aufzeichnung der Topologie-Daten über eine eigene XML-Datei. Diese hat eine bessere Strukturierung und beinhaltet nur relevante Informationen.

Vorteil

- Bessere Strukturierung der Topologie-Informationen
- Beinhaltet nur relevante Informationen, die für die Visualisierung eines OMNeT++ Netzwerkes nötig sind.

Nachteil

- Implementierungsaufwand
- Eine Änderung der OMNeT++ Version kann die Implementierung unbrauchbar machen

Wahl der Methode für die Erfassung der Topologie-Daten

Die Implementierung einer eigenen XML-Datei ist eine Möglichkeit, die Struktur der aufgezeichneten Topologie-Daten besser zu gestalten. Da aber die OMNeT++ Umgebung stetig weiter entwickelt wird, könnte eine Versionsänderung die Implementierung dieser Methode unbrauchbar machen. Das bereits integrierte XML-Format von OMNeT++ hingegen wird stetig mit gepflegt, verursacht deutlich weniger Aufwand und erfüllt die Anforderungen, die in Abschnitt 3.1 definiert wurden. Somit ist es die ideale Variante für den Export der Topologie-Daten. Weitere Informationen zum Aufbau des XML-Files in Kapitel 5 Abschnitt 5.1.1.

4.1.2. Metrik-Daten

Ein weiteres Ziel dieser Arbeit ist es, die Daten der OMNeT++ Simulation zu erfassen und diese auf dem Topologie Analyzer abzubilden. Um an die Simulationsdaten zu kommen, bietet OMNeT++ zwei bereits integrierte Ansätze (Scalar-/Vector- und Eventlog-Datei). Diese werden zunächst in diesem Abschnitt vorgestellt, dann werden die Vor- und Nachteile erläutert. Am Ende des Abschnitts wird entschieden, welcher dieser Ansätze eingesetzt wird.

Scalar-/Vector-Datei
Die Vector-Datei ist eine Ansammlung von Werten, die in den Modulen oder Channels aufgezeichnet werden. Die Vector-Outputs beinhalten Informationen, wie die Ende-zu-Ende-Latenz, die Bufferlänge, die Bufferungszeit, die Linkauslastung und mehr. Diese Informationen ermöglichen dem Nutzer, zu einem genauen Zeitpunkt der Simulation einen Wert zu lesen und somit die Ursache für ein Problem im Netzwerk zu finden, beispielsweise den Latenzwert von einem virtuellen Link zur einer bestimmten Simulationszeit zu lesen. Die Scalar-Datei ist eine Zusammenfassung von Ergebnissen, die während der Simulation berechnet wurden. Diese beinhaltet Informationen, wie Zähler, minimale und maximale Werte von Metriken (z.B. minimaler und maximaler Wert der Latenz eines virtuellen Links). Aus der Scalar-Datei können somit Informationen entnommen werden, die über den gesamten Simulationsablauf entstanden sind. In der Vector- und Scalar-Datei werden die Metrikinformationen anhand eines Header am Anfang der Datei definiert. Dieser beinhaltet die IDs der jeweiligen Metrik, den Namen des Moduls und den Metriktyp. Dank dieser Daten ist das Parsen der darauf folgenden Informationen (Metrikwerte und zugehöriger Simulationszeitpunkt) ideal. Auf Basis der Scalar-Datei kann eingegrenzt werden, auf welchem Modul hohe Latenzen entstanden sind. Dies wird dank der Minimal-, der Maximal- und der Durchschnittswerte der Latenz möglich. Die Ergebnisdaten können über zwei Varianten aufgezeichnet werden:

1. Basierend auf dem Signal-Mechanismus unter Verwendung deklarierter Statistics
2. Direkt über den C++ Code unter Verwendung der Simulationsbibliothek

Beide Files können über die .ini Datei an- und ausgeschaltet und über Parameter gefiltert werden [vgl. Manual OMNeT++ Community, b, Abschnitt 12].

Vorteil:

- Bereits in OMNeT++ integriert
- Beinhaltet alle statistischen Ergebnisse vom Netzwerk
- Ursachenfindung anhand der Metriken möglich
- Das Parsen der Daten ist anhand der ID ideal

Nachteil:

- Dateigröße kann nach einer längeren Simulationsdauer zu groß werden
- Die daraus resultierenden Dateien sind in zwei Formaten geschrieben, sodass zwei Parser benötigt werden

Eventlog-Datei

Die Eventlog-Datei zeichnet alle auftretenden Events der Simulation auf. Dazu zählen das Erstellen und Löschen von Gates, Modulen und Connections und das Senden von Nachrichten. Mithilfe des Eventlogs kann eine Simulation komplett Schritt für Schritt erneut konstruiert werden. Dabei wird der Schwerpunkt auf das Verhalten des Netzwerkes anstatt auf statistische Ergebnisse gelegt. Jedes Event wird mit einem Zeitstempel versehen und am Ende der Simulation kann anhand eines „Identifier“ bestimmt werden, um welches Event es sich handelt. Eine Liste mit allen Identifier befindet sich im OMNeT Manual (siehe [OMNeT++ Community, b, Abschnitt 27]). Durch das Speichern aller auftretenden Events kann das Eventlog sehr umfangreich werden, sodass die Datei mehrere Hundert Mega Byte, je nach Simulationsdauer sogar Giga Byte, groß werden kann. Das Aufzeichnen der Eventlog-Datei kann in der .ini-Datei an- und ausgeschaltet werden [vgl. Manual OMNeT++ Community, b, Abschnitt 13].

Vorteil:

- Bereits in der OMNeT++ Umgebung integriert
- Beinhaltet alle Informationen zum Aufbau und Verlauf des Netzwerkes während der Simulationszeit
- Ursachenfindung anhand der großen Anzahl an Informationen möglich

Nachteil:

- Nicht alle Metrik-Informationen sind in der Eventlog-Datei enthalten
- Durch die Aufzeichnung der Eventlog-Datei steigt die Simulationszeit drastisch an
- Die Dateigröße der resultierenden Datei steigt je nach Simulationsdauer stark
- Anhand der Dateigröße ist das Parsen der Informationen sehr zeitaufwendig
- Beinhaltet keine statistsichen Ergebnisse (Minimal-, Maximal-, Durchschnittswerte)

Wahl der Methode zur Erfassung der Metrik-Daten
Da alle Zeiten und Werte in der Scalar- und Vector-Datei nacheinander aufgezeichnet werden und anhand der ID vom Header leicht wiedergefunden werden können, eignen sich diese Dateien ideal zum Parsen der Informationen. Die Vector-Datei enthält alle Metriken (Metrikwerte und zugehörige Simulationszeitpunkte). Die Scalar-Datei enthält die zusammengefassten Metrikwerte, die über die gesamte Simulationszeit entstanden sind. Die Eventlog-Datei hingegen zeichnet sämtliche Daten während einer Simulation auf. Dementsprechend enthält diese nicht relevante Daten, wie z.B. das Erzeugen und Löschen von Modulen, Gates und Connections etc. Viele der Metriken können der Eventlog-Datei nicht entnommen werden, wie z.B. die Linkauslastung, die Bufferlänge zu bestimmten Zeiten, etc. Aus diesen Gründen ist die Aufzeichnung der Metrikdaten über die Eventlog-Datei nicht geeignet. Somit ist die Scalar- und Vector-Datei für den gewählten Ansatz zur Datenerfassung, die Darstellung von Metriken in einer Topologie, die geeignete Variante. Weitere Informationen zum Aufbau der Scalar- und Vector-Datei in Kapitel 5 Abschnitt 5.1.2 und 5.1.3.

4.2. Konzept des Topology Analyzer

In diesem Abschnitt wird das Konzept des Topology Analyzer entwickelt. Da dieser als Plug-In in die OMNeT++ Umgebung integriert wird, befasst sich der erste Abschnitt mit den Einstellungen, Extensionpoints und Dependencies des Plug-Ins. Dann wird der GUI-Entwurf des Topologie Analyzer vorgestellt und erläutert. Zuletzt wird die Darstellung der Metriken erläutert.

4.2.1. Plu-In-Einstellungen, Extensionpoints und Dependencies

Das Plug-In soll für den Nutzer einfach bedienbar sein. Um diese Anforderung zu gewährleisten, muss zunächst entschieden werden, wo das Plug-In auf der OMNeT++ Umgebung eingebunden werden soll. Dies wird über die Extensionpoints realisiert. Die Eclipse-Umgebung enthält verschiedene Optionen, mit denen das Plug-In gestartet werden kann. Es ist möglich, das Plug-In über die Toolbar, mit einem Rechtsklick über das Kontextmenü oder über die Menüleiste der OMNeT++ Umgebung zu starten. Für den Topology Analyzer wurde die Menüleiste gewählt, da diese am komfortabelsten für den Nutzer ist. Das Einbinden des Topology Analyzer auf der Menüleiste erfolgt über den Extensionpoint *org.eclipse.ui.menus*. Um festlegen zu können, welche Klasse beim Start ausgeführt werden soll, wird zusätzlich der extensionpoint *org.eclipse.ui.commands* benötigt. Über einen Handler wird bestimmt, welche Klasse beim Start genutzt werden soll. Zusätzlich wurde der Extensionpoint *org.eclipse.ui.bindings* verwendet. Diese ermöglicht das Starten der Applikation über ein Tastenkürzel der Tastatur.

Für die Erstellung der Statistiken im Topologie Analyzer wird das swt-xy-graph Plug-In eingesetzt [Eclipse Nebula Visualization]. Dieses wird als Dependencie in der Plug-In-Einstellung eingetragen. Weitere Informationen zum swt-xy-graph in Abschnitt 4.2.3. Des Weiteren muss die GUI-Entwicklungsbibliothek ausgewählt werden. Hierfür gibt es verschiedene Möglichkeiten, wie z.B. AWT, SWT oder Swing. Die Eclipse-Umgebung verfügt über eine integrierte SWT-Bibliothek (siehe Abbildung 2.2 in Abschnitt 2.3.1) für die Entwicklung von GUIs. Diese verfügt über alle Funktionalitäten der GUI-Entwicklung und ist für die Plug-In-Entwicklung bereits eingebunden. Diese erfüllt die Anforderungen aus dem Abschnitt 3.3.1 und ist die geeignete Wahl für die Entwicklung von GUIs.

4.2.2. GUI-Entwurf

Für den Entwurf der GUI des Topologie Analyzer werden drei Ansichten entwickelt:

1. Viewpoint View: zur Auswahl des Viewpoints, durch den die dargestellten Daten eingegrenzt werden sollen.

2. Data Selection View: zur Auswahl der Datei, in der die Topologie- und Metrik-Daten enthalten sind.

3. Topology View: In diesem wird das OMNeT++ Netzwerk mit den entsprechenden Metriken, die unter dem Abschnitt 2.2 erwähnt wurden, visualisiert.

Im Folgenden werden die drei Ansichten als Designskizze konzeptioniert und beschrieben.

Viewpoint View

Die *Viewpoint View* wird beim Start des Plug-Ins aufgerufen. In dieser Ansicht muss der Nutzer einen Viewpoint auswählen, mit dem bestimmte Funktionen und Ansichten des Programms ausgewählt werden. Die Ansicht ist schlicht konzeptioniert und hat lediglich eine Menüleiste mit einem Viewpoint-Button und einer Fläche, auf der das CoRE-Logo visualisiert wird. Mit einem Klick auf den Viewpoint-Button wird die Auswahl der Viewpoints ermöglicht. Jeder Viewpoint ist mit einem Tooltip[1] versehen, über den der Nutzer weitere Informationen erhält. Nach der Auswahl des Viewpoints wird diese Ansicht geschlossen und die nächste Ansicht wird erzeugt. Weiteres zu den Viewpoints findet sich in Abschnitt 4.2.3. In Abbildung 4.2 ist der Aufbau der *Viewpoint View* zu sehen.

[1] Ein Tooltip ist ein kleines Informationsfenster (Pop-up-Fenster). Dies erscheint nach einigen Sekunden, wenn der Mauszeiger auf ein Element draufzeigt. [TechTerms]

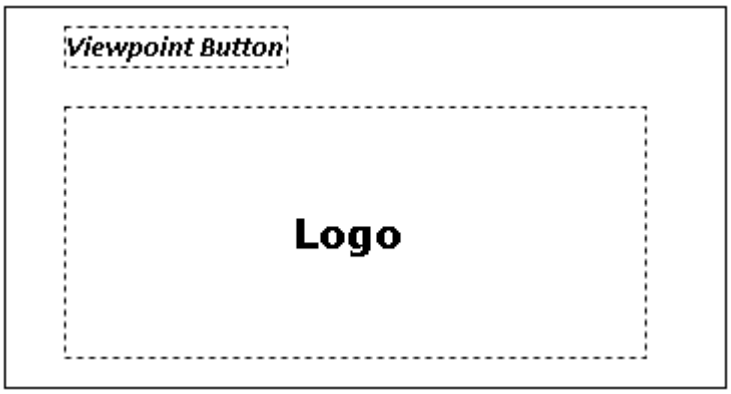


Abbildung 4.2.: Viewpoint View

Data Selection View

Die nächste Ansicht ist die *Data Selection View*. Diese unterscheidet sich strukturell nicht von der Viewpointansicht. Die Ansicht ist ebenfalls schlicht konzeptioniert und hat nur eine Fläche mit dem CoRE-Logo und eine Menüleiste mit einem Open-Button. Über den Open-Button kann der Nutzer die Informationen in Form einer XML- (.xml), einer Scalar- (.sca) und einer Vector- (.vec) Datei einlesen. Das Einlesen der Dateien läuft wie folgt ab: Zuerst wird die Topologie-Datei (XML-Datei), dann werden die dazugehörigen Metrik-Daten (Vector- und Scalar-Datei) eingelesen. Der Nutzer muss selbst darauf achten, die richtigen Informationen einzulesen, da keine Fehlermeldung ausgegeben wird, falls dies nicht der Fall ist. Da das Einlesen der Topologie- und Metrik-Informationen dauern kann, wird eine zusätzliche Ansicht beim Einlesen erzeugt. In dieser Ansicht wird „Loading, please wait“ angezeigt, was dem Nutzer signalisieren soll, dass die Informationen gerade eingelesen werden. Am Ende des Einlesens wird diese zusätzliche Ansicht wieder entfernt. In Abbildung 4.3 ist der Aufbau der *Data Selection View* zu sehen.

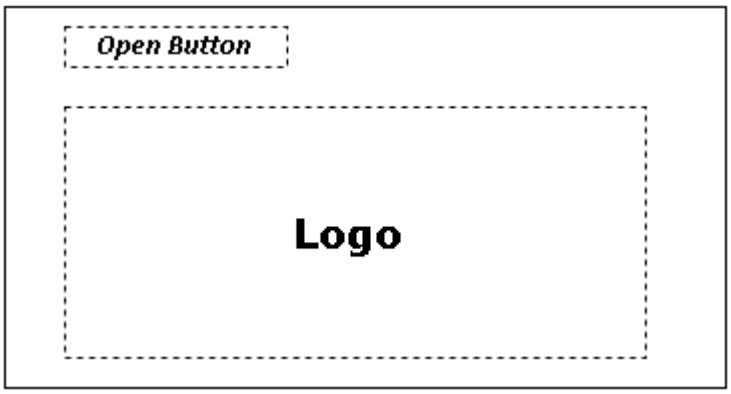


Abbildung 4.3.: Data Selection View

Topology View

Die *Topology View* ist die letzte der drei Ansichten. Nach dem Einlesen der Topologie- und Metrik-Informationen wird diese Ansicht erzeugt. Die Topologie des Netzwerkes wird in dieser Ansicht visualisiert (Topologie-Anzeigefläche), einige Metriken werden in dieser dargestellt. Über diese Ansicht kann der Nutzer die Metriken im Netzwerk gezielt beobachten (Auswahl-Button und Anzeigefläche). Dies erleichtert die Ursachenfindungen. In Abbildung 4.4 ist der Aufbau der *Topology View* zu sehen.

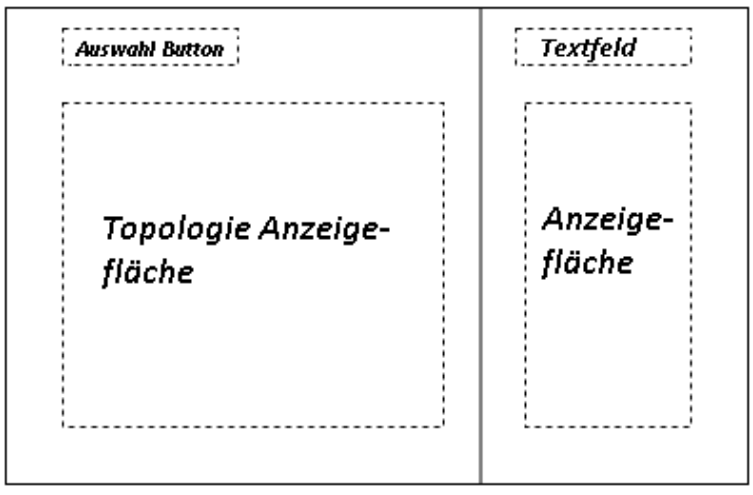


Abbildung 4.4.: Topology View

4.2.3. Darstellung der Metriken

In diesem Abschnitt wird zunächst anhand einer Visualisierungssoftware des Deutschen Forschungsnetzes (DFN) beschrieben [Customer Network Management, a], wie die Visualisierung von Netzwerken (Topologien) und Metriken dargestellt werden. Anschließend wird die Visualisierung von Topologie und Metriken mit dem Topology Analyzer beschrieben. Dann werden die Statistiken des Topologie Analyzers erläutert und der Plug-In swt-xy-graph, der als dependencie eingebunden wird, vorgestellt. Zuletzt werden Viewpoints erläutert.

Darstellungsvarianten der Metriken

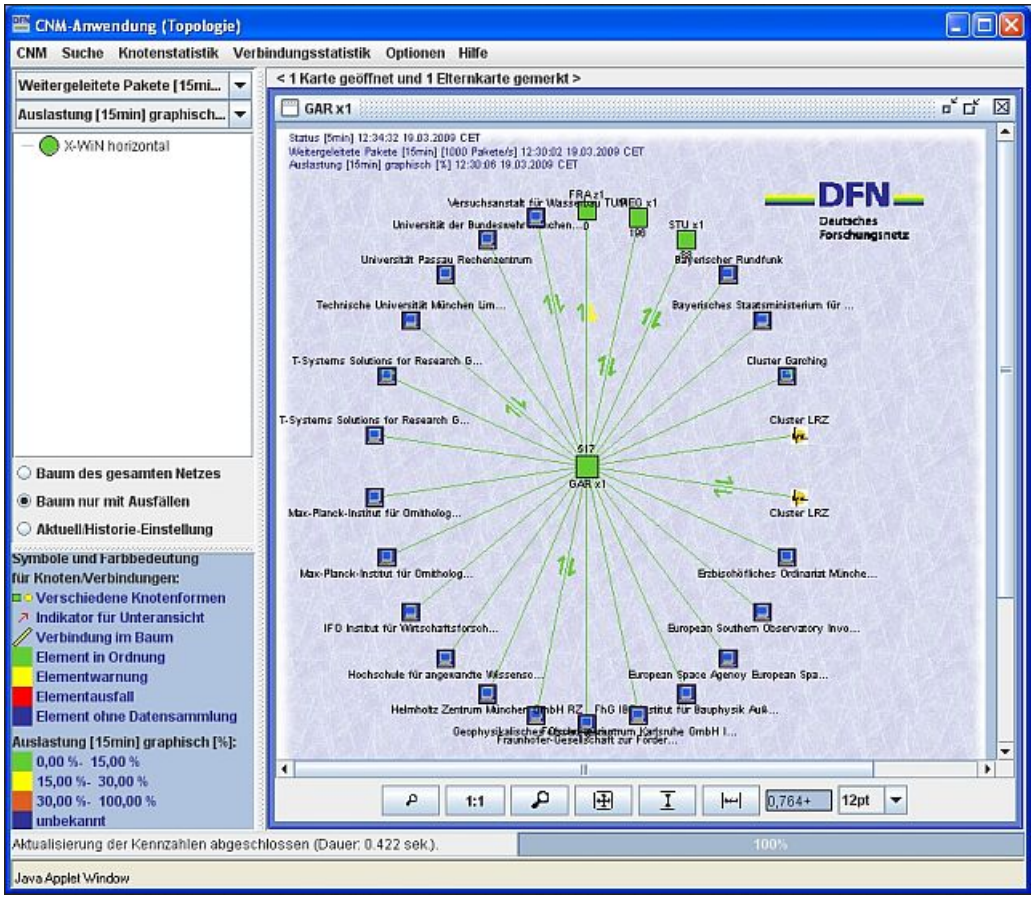


Abbildung 4.5.: Topologie MainView der CNM-Anwendung

In Abbildung 4.5 ist die Customer Network Management (CNM) Anwendung des Deutschen Forschungsnetzes (DFN) abgebildet. In dieser wird die Topologieansicht des Deutschen Forschungsnetzes dargestellt. Diese besteht aus einem zentralen Router (GAR x1), der mit diversen Netzwerken an verschiedenen Standorten verbunden ist, wie z.B. dem Netzwerk der Technischen Universität München. In dieser Ansicht wird die Auslastung der Netzwerke visualisiert. Anhand der Verbindungen wird farblich dargestellt, wie stark die einzelnen Netzwerke ausgelastet sind. In Abbildung 4.5 (unten links) ist zu erkennen, ab welcher Auslastung (Prozentual) eine Verbindung ihre Farbe ändert [Customer Network Management, a].

Des Weiteren bietet diese Software die Möglichkeit, Metriken über Statistiken darzustellen. Die Abbildung 4.6 zeigt eine Wochenstatistik des Durchsatzes des zentralen Routers des DFN (siehe [Customer Network Management, b, User Manual]).

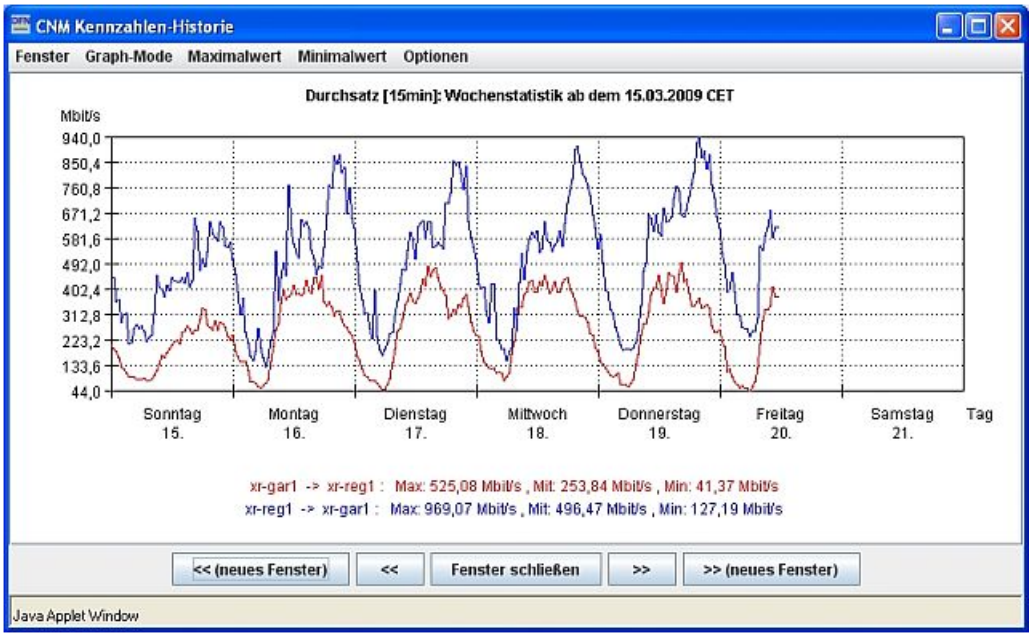


Abbildung 4.6.: Wochenstatistik

Darstellung der Metriken im Topology Analyzer

Im Folgenden wird die Darstellung der Metriken im Topology Analyzer erläutert. Das Programm aus Abbildung 4.5 diente als Vorlage für die Darstellung der Linkauslastung. Außerdem inspirierte das Beispiel (siehe Abbildung 4.6) dazu, Statistiken für Metriken im Topology Analyzer anzuwenden.

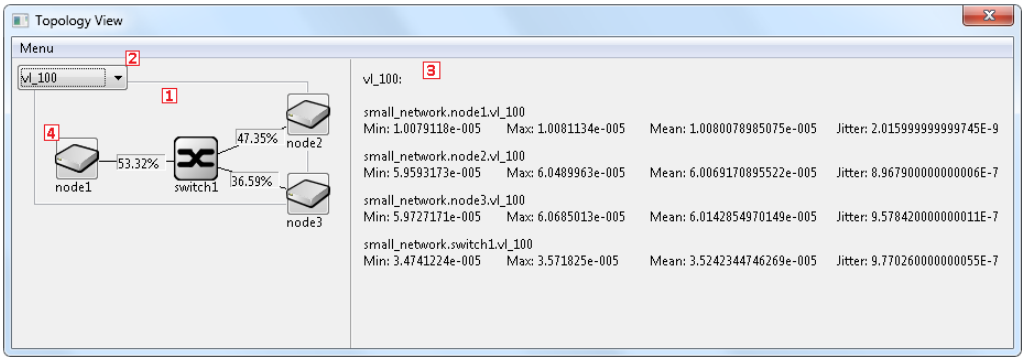


Abbildung 4.7.: Topologieansicht des small_network

Abbildung 4.7 zeigt das OMNeT++ Netzwerk (*small_network*) aus Abbildung 4.1 im Topology Analyzers. Folgende Metriken werden im Topology Analyzer dargestellt:

- Latenz sowie ihr Minimal-, Maximal- und Mittelwert
- Jitter
- Bufferstände der einzelnen Nachrichtentypen (TT, avbA/B, etc.)
- Paketverlustrate im Buffer
- Linkauslastung der physikalischen Verbindung
- Bandbreite der physikalischen Verbindung

Im Topology View-Fenster wird auf der linken Seite (siehe [1] in der Abbildung 4.7) das OMNeT++ Netzwerk dargestellt. Dieses wurde mit Hilfe der eingelesenen Topologie-Daten (XML Datei) visualisiert. Die Topologie-Darstellung im Topology Analyzer ist ein Abbild des OMNeT++ Netzwerkes aus Abbildung 4.1.

Auf den Verbindungen zwischen den Modulen wird die Bandbreite und Linkauslastung dargestellt. Über einen Rechtsklick auf die Verbindung kann ausgewählt werden, welche der Metriken (Bandbreite oder Linkauslastung) dargestellt werden soll.

Außerdem ist auf der linken Seite ein Auswahl-Button zu sehen (siehe [2] der Abbildung 4.7), über den auf der rechten Seite (siehe [3] der Abbildung 4.7) der Jitter und der Minimal-, Maximal- und Mittelwert der Latenz dargestellt werden. Anhand der Zeiten kann der Pfad, den eine Nachricht durchlaufen hat, nachverfolgt werden. Wie z.B in Abbildung 4.7 zu sehen ist, verlief die Nachricht von *node1* (10,08µs) über den *switch1* (34,7µs) zu *node2* (59,6µs) und *node3* (59,7µs). Außerdem kann anhand der Zeiten erkannt werden, ob es an einem der Module eine Zeitverzögerung gab.

Mit einem Rechtsklick auf ein Modul (siehe [4] der Abbildung 4.7) können Statistiken ausgewählt werden. Die Daten für die Statistiken werden der Vector-Datei entnommen. Folgende Metriken werden als Statistiken dargestellt:

- Bufferstände der einzelnen Nachrichtentypen (TT, avbA/B, etc.)
- Latenzen der einzelnen Nachrichtentypen
- Paketverlustrate im Buffer (engl. dropped packets)

Statistiken dienen der genaueren Betrachtung von großen Datenwerten. Die oben genannten Metriken und Funktionen können anhand von Viewpoints eingegrenzt werden. In den folgenden Abschnitten werden die Erzeugung von Statistiken und der Einsatz von Viewpoints erläutert.

Konzept der Statistiken

In diesem Abschnitt wird die Darstellung der Statistik von Metriken des Topology Analyzer erläutert. Die Statistiken sollen dem Nutzer ermöglichen, Metrikwerte zu genaueren Simulationszeiten zu erfassen, sodass mögliche Verzögerungszeiten mit den exakten Simulationszeiten ausfindig gemacht werden können. Der Topology Analyzer verwendet zur Erzeugung von Statistiken Funktionen des swt-xy-graph, dieser wird als Dependencie in der Plug-In-Einstellung eingebunden.

Der swt-xy-graph ist ein Plug-In der Eclipse Nebula, gehört zu den Visualization Widgets ([Eclipse Nebula Visualization]) und dient lediglich zur Visualisierung von Statistiken.

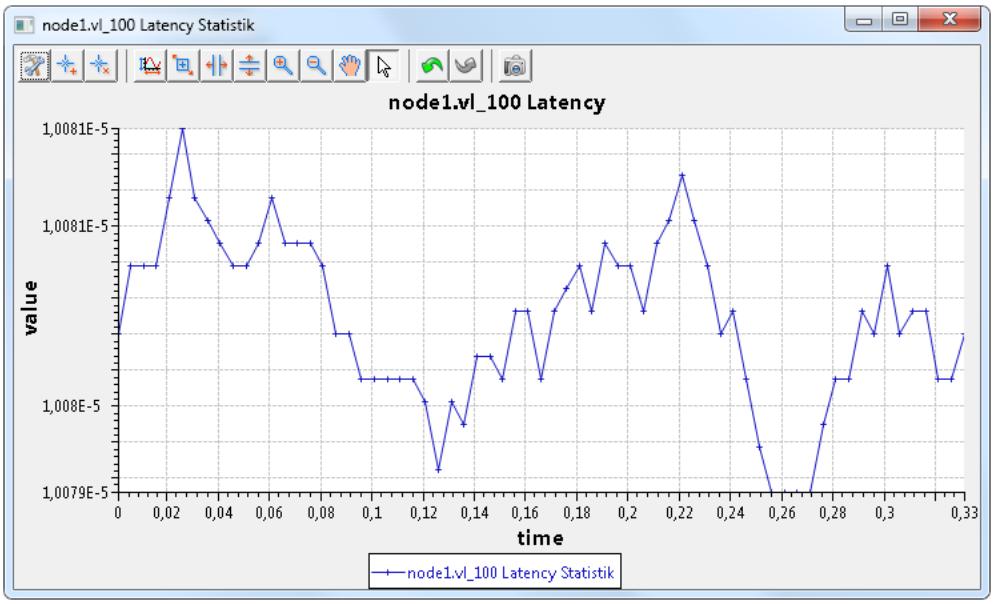


Abbildung 4.8.: Latenz-Statistik

In Abbildung 4.8 ist die Latenz-Statistik des vl_100 von node1 aus Abbildung 4.7 zu sehen. Die Metrikwerte und Simulationszeiten werden der Vector-Datei entnommen und in der Statistik abgebildet. In der Latenz-Statistik ist auch der Minimal- und Maximalwert der Latenz aus Abbildung 4.7 zu erkennen. So ist der Nutzer in der Lage, anhand der Minimal-, Maximal- und Mittelwerte mögliche hohe Verzögerungszeiten zu finden und

auf Basis der Statistiken genauere Analysen vorzunehmen. Der swt-xy-Graph bietet des Weiteren verschiedene Funktionen, die eine bessere Darstellung der Werte ermöglichen, wie z.B. die Zoom-Funktion, das Setzen von Annotationen, die Möglichkeit, Snapshots von der Statistik zu erstellen, etc.

Viewpoint

Ein Viewpoint beschreibt eine Perspektive bzw. eine Sicht, d.h. durch Anwendung von Viewpoints sollen verschiedene Sichtweisen im Topology Analyzer beschrieben werden. Über die *Viewpoint View* kann ein Viewpoint im Topology Analyzer ausgewäht werden (siehe Abbildung 4.2 aus Abschnitt 4.2.2). Der Topology Analyzer verwendet zwei Viewpoints:

- Master Viewpoint: Verfügt über alle Metriken und Funktionen, die im vorherigen Abschnitt beschrieben worden sind.
- Average Viewpoint: Verfügt lediglich über die Metriken aus der Scalar-Datei. Diese sind der Minimal-, der Maximal- und der Mittelwert der Latenz, der Jitter, die Bandbreite und die Linkauslastung der physikalischen Verbindung.

Anhand der Viewpoints sollen Funktionalitäten und Metriken eingrenzen werden, sodass nur bestimmte Sichten zur Verfügung stehen. Genaueres zur Implementierung der Viewpoints in Kapitel 5.

5. Implementation

In diesem Kapitel wird die Implementierung des Topology Analyzers beschrieben. Zunächst wird der Aufbau der Datenformate (XML-, Scalar- und Vector-Datei) erläutert. Danach folgt die Implementierung des Eclipse Plug-In Projekts. Dabei wird die Programmarchitektur beschrieben, die einzelnen Klassen der Packages werden erläutert und zuletzt wird die Plug-In-Konfiguration dargestellt.

5.1. Aufbau der Datenformate

In diesem Abschnitt wird der Aufbau der Datenformate beschrieben. Zunächst wird der Aufbau der XML-Datei erläutert. Dann wird der Aufbau der Scalar- und der Vector-Datei erläutert.

5.1.1. XML-Datei

Im Listing 4.1 auf Seite 18 wurde der Aufbau des small_network.ned beschrieben. Im folgenden Listing wird die exportierte XML-Datei von der small_network.ned gezeigt:

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2
3  <compound-module name="small_network">
4     <comment locid="banner" content="//&#10;//
5  ___Small_sample_network_with_three_hosts_and_one_switch_that_shows_how_to_&#10;
6  ___//configure_the_TTEthernet_Model.
7  ___Configuration_in_omnetpp.ini&#10;//
8  ___-_Uses_5ms_Cycletime&#10;//_-_80ns_Tick_length&#10;//&#10;"/>
9
10    <!-- Kommentar: Diese Parameter verändern das Aussehen der Simulation -->
11    <parameters is-implicit="true">
12      <property is-implicit="true" name="isNetwork"/>
13      <property name="display">
14        <property-key>
15          <literal type="string" text="&quot;bgb=,,white&quot;" value="bgb=,,white"/>
```

```
16        </property-key>
17      </property>
18    </parameters>
19
20    <submodules>
21      <!-- Kommentar: Deklaration von node1 -->
22      <submodule name="node1" type="Node1">
23        <comment locid="trailing" content="&#10;&#10;"/>
24        <parameters is-implicit="true">
25          <property name="display">
26            <property-key>
27              <literal type="string" text="&quot;p=39,70&quot;" value="p=39,70"/>
28            </property-key>
29          </property>
30        </parameters>
31      </submodule>
32
33      <!-- Kommentar: Deklaration von switch1 -->
34      <submodule name="switch1" type="Switch1">
35        <parameters>
36        <property name="display">
37          <property-key>
38            <literal type="string" text="&quot;p=147,70&quot;" value="p=147,70"/>
39          </property-key>
40        </property>
41        </parameters>
42        <gates>
43          <gate name="ethg" is-vector="true" vector-size="3"/>
44        </gates>
45      </submodule>
46      ...
47    </submodules>
48
49    <connections>
50      <!-- Kommentar: Definierung der Verbindung zwischen node1 und switch1 -->
51      <connection src-module="node1" src-gate="ethg" dest-module="switch1"
52      dest-gate="ethg" dest-gate-index="0" type="Eth100M" is-bidirectional="true">
53        <comment locid="banner" content="//&#10;//_The_example_uses_a
54 ......simple_star_topology&#10;//&#10;"/>
55        <parameters is-implicit="true">
56          <param name="length" value="20m"/>
57        </parameters>
58      </connection>
59      ...
60    </connections>
61  </compound-module>
```

Listing 5.1: XML-Format von small_network.ned

Wie im Listing 5.1 zu erkennen ist, werden sämtliche Netzwerk-Informationen aus dem Listing 4.1 in der XML-Form beschrieben. In den Zeilen 10 bis 18 werden die Netzwerk-Darstellungsinformationen beschrieben, die das Aussehen in der NED-Darstellung und in der OMNeT++ Simulation beeinflussen. In den Zeilen 20 bis 47 werden die Submodule, die im Netzwerk definiert wurden, beschrieben. Unter diesen Informationen werden die Namen und die Typen der Module (Zeilen 22 und 34), die Positionen der Module (Zeilen 27 und 38) beschrieben. Zusätzlich zu den Switches werden der Name des Gates und die Vektorgröße des Gates beschrieben (Zeile 43). In den Zeilen 49 bis 60 wird die Verbindung der Module beschrieben. Darunter sind die Informationen, welches das Source- und welches das Destination-Modul ist (Zeile 51) und um welchen Typ der Verbindung es sich handelt (Zeile 52).

5.1.2. Vector-Datei

Im Folgenden wird ein Teilausschnitt der Ergebnisse der Vector-Datei aus dem small_network Beispiel der Simulation dargestellt.

```
1  ...
2  vector 2  small_network.node1.avbCTC  latency:vector  ETV
3  attr interpolationmode  none
4  attr source  messageAge(rxPk)
5  attr title  "End-to-end latency, vector"
6  attr unit  s
7  vector 3  small_network.node1.avbABuffer[0]  rxBytes:vector(packetBytes)
   ETV
8  attr interpolationmode  none
9  attr source  rxPk
10 attr title  "Received Bytes, vector(packetBytes)"
11 attr unit  B
12 vector 5  small_network.node1.avbABuffer[0]  rxLatency:vector  ETV
13 attr interpolationmode  none
14 attr source  messageAge(rxPk)
15 attr title  "End-to-end latency (received), vector"
16 attr unit  s
17 vector 8  small_network.node1.avbABuffer[0]  length:vector  ETV
18 attr interpolationmode  sample-hold
19 attr source  length
20 attr title  "Queue Length, vector"
21 attr unit  packets
22 ...
```

```
23 8        78683    0.17621847764    0
24 8        78735    0.176317920126   1
25 8        78750    0.176343120756   0
26 8        78789    0.176442963252   1
27 8        78798    0.176467763872   0
28 8        78850    0.176568006378   1
29 8        78858    0.176592406988   0
30 8        78904    0.176693049504   1
31 ...
```

Listing 5.2: Teil Ausschnitt der Vector-Datei

Die Vector-Datei hat zwei Typen von Zeilen [vgl. OMNeT++ Community, b, Abschnitt 12.3]:

- Die ersten Zeilen sind die Vector-Deklarationen (Vector Header, siehe Zeilen 2 bis 21 von Listing 5.2), diese beginnen mit dem Keyword „vector“ und beschreiben pro Vector eine Metrik eines Moduls. Die Vector-Deklaration ist wie folgt strukturiert:
  - Vector-ID
  - Name des Moduls
  - Name der Metrik
  - Weitere Zeilen mit Attributen, wie zum Beispiel Titel, Source und Einheit.
- Der zweite Typ der Zeilen erscheint, wenn alle Vector-Deklarationen definiert wurden (siehe Zeilen 23 bis 30 von Listing 5.2). Diese werden als Data Lines bezeichnet und sind wie folgt strukturiert:
  - Vector-ID
  - Event-Nummer
  - Simulationszeit
  - Vector-Wert

Zum Auslesen der Daten muss der Header der jeweiligen Metriken gespeichert werden. Anhand der Vector-ID können dann die eigentlichen Daten ausgelesen werden.

5.1.3. Scalar-Datei

Die Scalar-Datei ist gleich strukturiert wie die Vector-Datei. Im Folgenden wird ein Teilausschnitt der Ergebnisse der Scalar-Datei aus dem small_network Beispiel der Simulation gezeigt.

```
...
statistic small_network.node1.phy[0].inControl  rxLatency:stats
field count 1434
field mean 0.00025205597227058
field stddev 7.4026508991162e-006
field sum 0.361448264236
field sqrsum 9.1183720978917e-005
field min 9.86e-006
field max 0.000299084235
attr interpolationmode  none
attr source  messageAge(rxPk)
attr title  "End-to-end latency (received), stats"
attr unit  s
scalar small_network.node1.phy[0].mac   "simulated time"        0.328007680211
scalar small_network.node1.phy[0].mac   full-duplex     1
scalar small_network.node1.phy[0].mac   "frames/sec sent"       12057.644496177
scalar small_network.node1.phy[0].mac   "frames/sec rcvd"       4371.8488514584
scalar small_network.node1.phy[0].mac   "bits/sec sent"         70821561.205691
scalar small_network.node1.phy[0].mac   "bits/sec rcvd"         53020807.283575
scalar small_network.node1.phy[0].mac   "rx channel idle (%)"   46.699394389931
scalar small_network.node1.phy[0].mac   "rx channel utilization (%)"    53.300605610069
...
```

Listing 5.3: Teilausschnitt der Scalar-Datei

Die Scalar-Datei hat im Gegensatz zur Vector-Datei nur einen Typ von Zeilen und wird als Scalar-Deklaration bezeichnet (bzw. Scalar Header) [vgl. OMNeT++ Community, b, Abschnitt 12.3]. Diese beginnt mit dem Keyword „scalar“ und beschreibt jeweils ein Scalar einer Metrik eines Moduls. Diese ist wie folgt strukturiert:

- Name des Moduls
- Name der Metrik
- Scalar-Wert

Zum Auslesen der Scalar-Informationen müssen die Scalar-Header und der Scalar-Value abgespeichert werden.

5.2. Eclipse Plug-In Projekt

In diesem Abschnitt wird die Implementierung des Topology Analyzers beschrieben. Zunächst wird ein Überblick über die Programmarchitektur des Plug-Ins dargestellt. Im Folgenden wird dann genauer auf die einzelnen Klassen der Packages eingegangen. Zuletzt wird die Plug-In Konfiguration erläutert.

5.2.1. Programmarchitektur

Der Topology Analyzer besteht aus einem Plug-In Projekt, welches folgende Struktur aufweist:

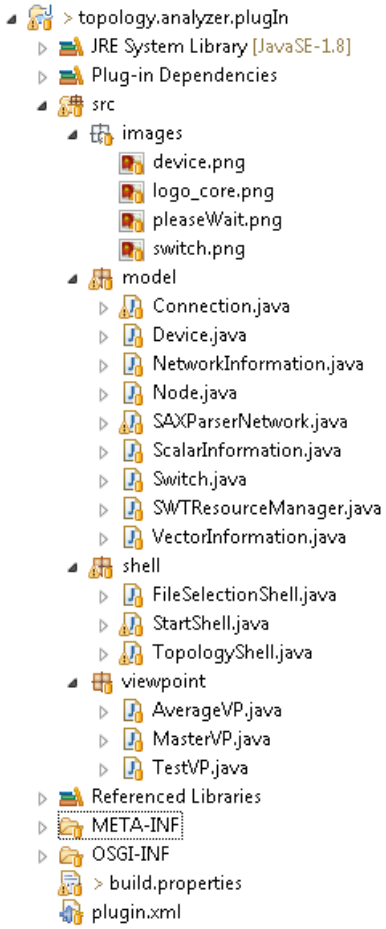


Abbildung 5.1.: Ecplise Plug-In Projekt Architektur

Das Plug-In Projekt wurde in Eclipse generiert und ist in vier Packages unterteilt: jeweils ein package für die Viewpoint, die Shell, das Model und die verwendeten Images des Topology Analyzers. Im Folgenden werden die Klassen der Packages mit einem Klassendiagramm vorgestellt und erläutert. Ein komplettes Klassendiagramm des Projekts befindet sich im seperaten Projektordner. Des Weiteren enthält das Plug-In Projekt einen META-INF Ordner und ein plugin.xml. Diese beinhalten die Konfigurationen des Plug-In.

5.2.2. Viewpoints

Dieses Package enthält die Viewpoint-Klassen des Topology Analyzers. Diese beeinflussen die Ansichten und Funktionalitäten für die Darstellung der Metriken. Der Topology Anaylzer verwendet zwei Viewpoint-Klassen, die im vorherigen Kapitel kurz erläutert wurden (siehe Abschnitt 4.2.3 auf Seite 33). Im Folgenden wird auf die Implementierung der *MasterVP.java* und *AverageVP.java* eingegangen.

MasterVP.java und AverageVP.java

Die Viewpoint-Klassen *MasterVP.java* und *AverageVP.java* werden in der *StartShell.java* (siehe Abschnitt 5.2.3) ausgewählt und nach der Auswahl instanziiert. Diese beinhalten Boolean-Attribute, die bestimmen, welche Ansichten und Funktionalitäten der *Node.java*, *Switch.java* und *Connection.java* Klasse verfügbar sind. In Listing 5.4 ist ein Ausschnitt der definierten Attribute des *MasterVP.java* zu sehen.

```
1  //Node Properties
2  Node.viewAvbAStatistik = true;  //Show MenuItem AvBAStatistik with the Buffer Statistik
3  Node.viewAvbBStatistik = true;  //Show MenuItem AvBBStatistik with the Buffer Statistik
4  ...
5  //switch Properties
6  Switch.viewAvbAStatistik = true;//Show MenuItem AvBAStatistik with the Buffer Statistik
7  Switch.viewAvbBStatistik = true;//Show MenuItem AvBBStatistik with the Buffer Statistik
8  ...
9  //Connection Properties
10 Connection.viewShowLinkauslastung = true;//show Menuitem Link Utility on Edge
11 Connection.viewShowBandbreite = true;//show Menuitem * on Edge
```

Listing 5.4: MasterVP.java - Ausschnitt der definierten Attribute

Die *AverageVP.java* Klasse ist genau wie die *MasterVP.java* Klasse aufgebaut und weist lediglich andere Attributwerte auf.

5.2.3. Shells

In diesem Package befinden sich die Shells, die als Ansichten des Topology Analyzer dienen. Diese Ansichten werden für die Auswahl der Viewpoints und der Ergebnisdaten und für die Darstellung der Topologie verwendet. Die Applikation hat drei Shells, die im Folgenden vorgestellt werden.

StartShell.java

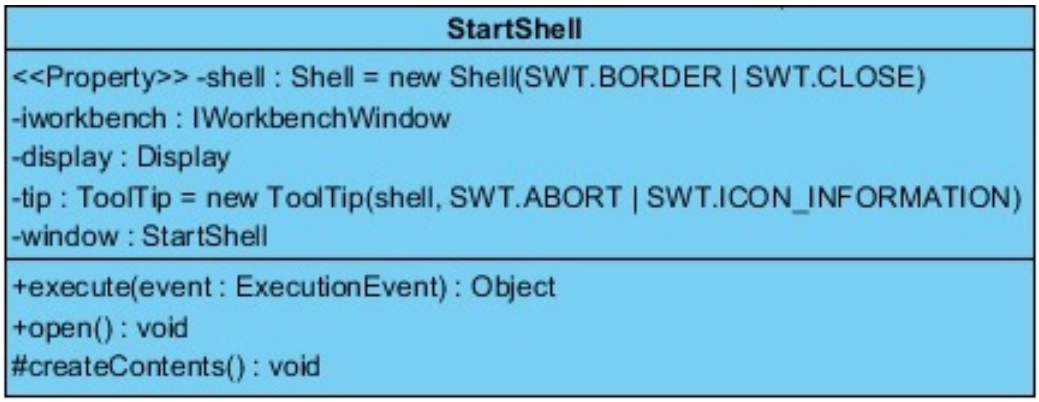


Abbildung 5.2.: Klassendiagramm - StartShell.java

Beim Start des Plug-Ins wird die *StartShell.java* mit der *execute* Methode ausgeführt. Diese Klasse erzeugt eine Shell für die Auswahl des Viewpoints (vgl. Abbildung 4.2). Nachdem der Nutzer einen Viewpoint ausgewählt hat, wird die *FileSelectionShell.java* Klasse aufgerufen und die *StartShell.java* wird geschlossen.

FileSelectionShell.java

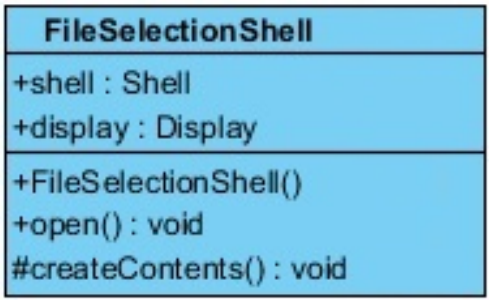


Abbildung 5.3.: Klassendiagramm - FileSelectionShell.java

Diese Klasse dient lediglich zur Auswahl der Topologie-Informationen (XML-Datei) und der Simulationsergebnis-Dateien (Scalar- und Vector-Datei). Nach der Auswahl der Dateien wird die *SAXParserNetwork.java* Klasse aufgerufen (siehe Abschnitt 5.2.4). Diese Ansicht bleibt nach der Auswahl weiterhin bestehen, um dem Nutzer die Möglichkeit zu geben, verschiedene Netzwerke mit verschiedenen Simulationsergebnis-Dateien einzulesen.

TopologyShell.java

Diese Ansicht ist die Hauptansicht des Topology Analyzer und visualisiert die eingelesenen Topologien und Metriken. Die Informationen werden zunächst von der *SAXParserNetwork.java* Klasse eingelesen (siehe Abschnitt 5.2.4) und in die nötigen Datenstrukturen gespeichert. Zusätzlich wird für jedes Modul, das das Netzwerk besitzt (Node, Switch, etc.), *Node.java* und *Switch.java* Klassen erzeugt. Diese werden dann in der Ansicht visualisiert und stehen in direkten Verbindung bei einer Aktion, d. h. bei einem Rechtsklick auf ein Modul werden die Funktionen der jeweiligen Klasse über einen Listener aufgerufen. Des Weiteren werden in der *TopologyShell.java* die Informationen aus der Scalar-Datei abgebildet.

```
1 public void menueForSimulation(List<Node> nodeListe, List<Switch> switchListe)
```

Listing 5.5: Methode für die Darstellung der Scalar-Informationen

Die Methode *menueForSimulation* (siehe Listing 5.5) erzeugt eine *Combobox* und bildet die statistischen Informationen der Scalar-Datei für jede existierende virtuelle Leitung, avbBuffer und app etc. ab. Nach der Auswahl in der Combobox werden dann der Jitter, die Minimal-, Maximal- und Durchschnittswerte der Latenz, die über die gesamte Simulation errechnet wurden, in der Topology View angezeigt (siehe Abbildung 4.4). Diese Informationen werden zunächst über den *SAXParserNetwork.java* ausgelesen und in den Datenstrukturen abgespeichert, sodass diese für die *Combobox* verwendet werden können. Diese Ansicht kann von der *FileSelectionShell.java* beliebig oft mit unterschiedlichen Netzwerken und Ergebnisdateien aufgerufen werden, sodass der Nutzer die Möglichkeit hat, zwei oder mehr Netzwerke miteinander zu vergleichen.

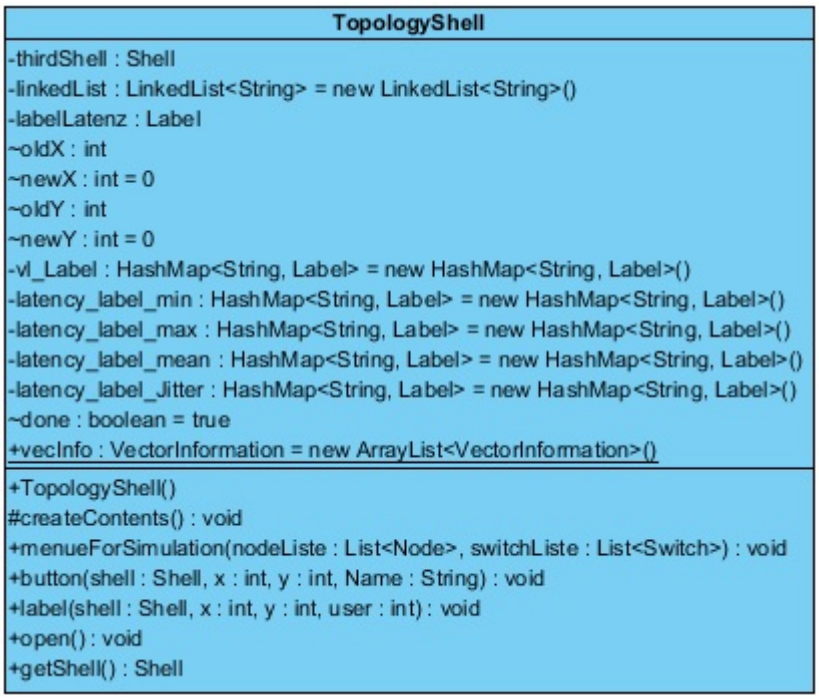


Abbildung 5.4.: Klassendiagramm - TopologyShell.java

5.2.4. Model

Die Klassen aus dem Package *Model* werden zum Auslesen der Topologie- und der Ergebnisdateien (XML-, Scalar- und Vector-Dateien), zum Abspeichern der eingelesenen Informationen in die Datenstrukturen, zum Erzeugen der Module und für die Funktionalitäten der Module verwendet. Die *SAXParserNetwork.java* Klasse liest die Informationen ein und speichert diese in den nötigen Datenstrukturen ab. Über die *Node.java* und *Switch.java* Klassen werden die Module erzeugt. Über die Klasse *Connection.java* wird die Verbindung der Module erzeugt. Im Folgenden wird auf diese Klassen genauer eingegangen. Alle anderen Klassen, die unter Abbildung 5.1 gelistet sind, dienen als Datenstrukturen für die Metrik- und Topologie-Informationen.

SAXParserNetwork.java

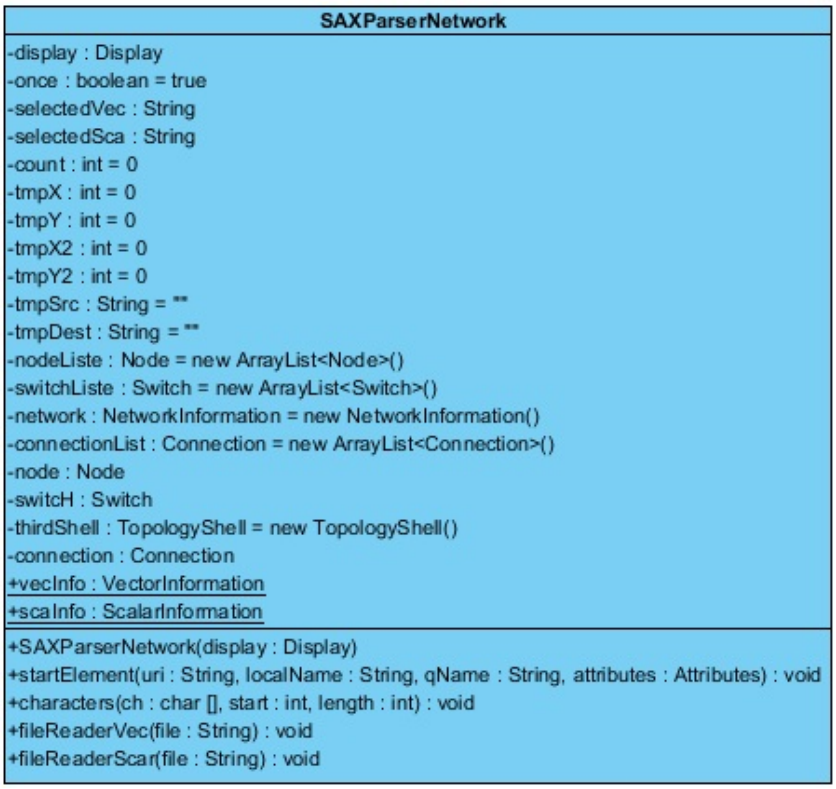


Abbildung 5.5.: Klassendiagramm - SAXParserNetwork.java

Die *SAXParserNetwork.java* Klasse wird für das Auslesen der Topologie- und der Ergebnis-Informationen (XML, Scalar und Vector) verwendet. Die Klasse basiert auf den Simple API for XML (SAX) Projekt und ist ein Open Source-Projekt für das Auslesen der XML-Informationen. Die Funktionalitäten, die für das Auslesen der XML-Informationen verwendet wurden, wurden von diesem Projekt übernommen [David Megginson].

Die Methode *startElement* (siehe Klassendiagramm in Abbildung 5.5) enthält eine Switch-Case, welche die Knoten-Elemente der XML-Datei durchläuft und alle wichtigen Informationen ausliest und speichert. Die Switch-Case wurde auf der XML-Struktur, das von OMNeT++ erzeugt wird, aufgebaut. Der Aufbau der Topologie des Netzwerkes in der Topology-View erfolgt während des Auslesens.

Die Methoden *fileReaderVec* und *fileReaderScar* (siehe Abbildung 5.5) werden für das Auslesen der Vector- und Scalar-Dateien verwendet. Das Auslesen der Dateien erfolgt zeilenweise und speichert die benötigten Header-Informationen der Scalar- und Vector-Datei (siehe Abschnitt 5.1.2 und 5.1.3). Anhand der ID werden die Vector-Daten in jeweils einem Array für Zeit und Value gespeichert. Dies kann je nach Dateigröße Zeit in Anspruch nehmen und wird aus diesem Grund einmalig ausgelesen und in jeweils einer Datenstruktur abgespeichert.

Node.java und Switch.java

Die *Node.java* und *Switch.java* Klassen representieren einen Node und einen Switch aus dem OMNeT++ Netzwerk. Für jeden existierenden Node und Switch im Netzwerk wird jeweils eine Klasse erzeugt. Die Statistiken der Metriken (Buffer, Latenz etc.) werden über diese Klasse erstellt. Nach der Auswahl der Viewpoints werden zunächst die Node- und Switch-Attribute gesetzt. Diese bestimmen, welche Funktionalitäten und Ansichten dem Nutzer zur Verfügung stehen. Über die *createBufferStat* Methode werden die Statistiken für die jeweiligen Buffer, die Anzahl der verlorenen Pakete und der Latenzen erstellt. Diese Methoden verwendet Funktionalitäten des swt-xy-Graph und erzeugt die Shell der Statistiken für alle Metriken mit einer großen Anzahl an vordefinierten Funktionalitäten (siehe Abschnitt 4.2.3 auf Seite 32). Das Klassendiagramm zu den beiden Klassen findet sich in Abbildung 5.6 und 5.7.

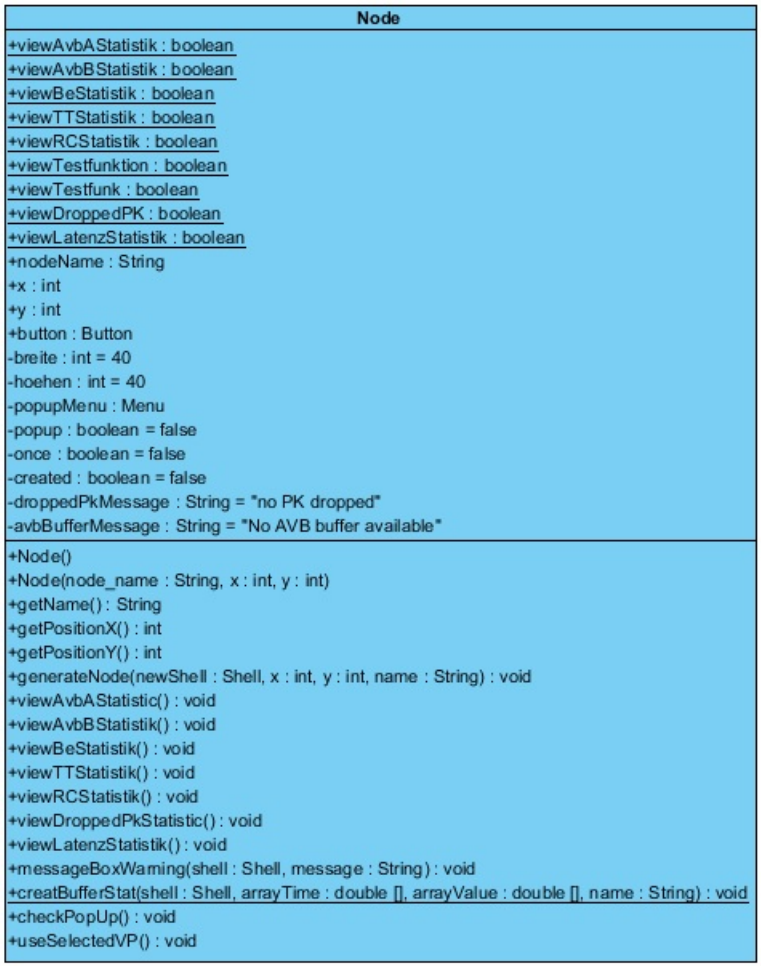


Abbildung 5.6.: Klassendiagram - Node.java

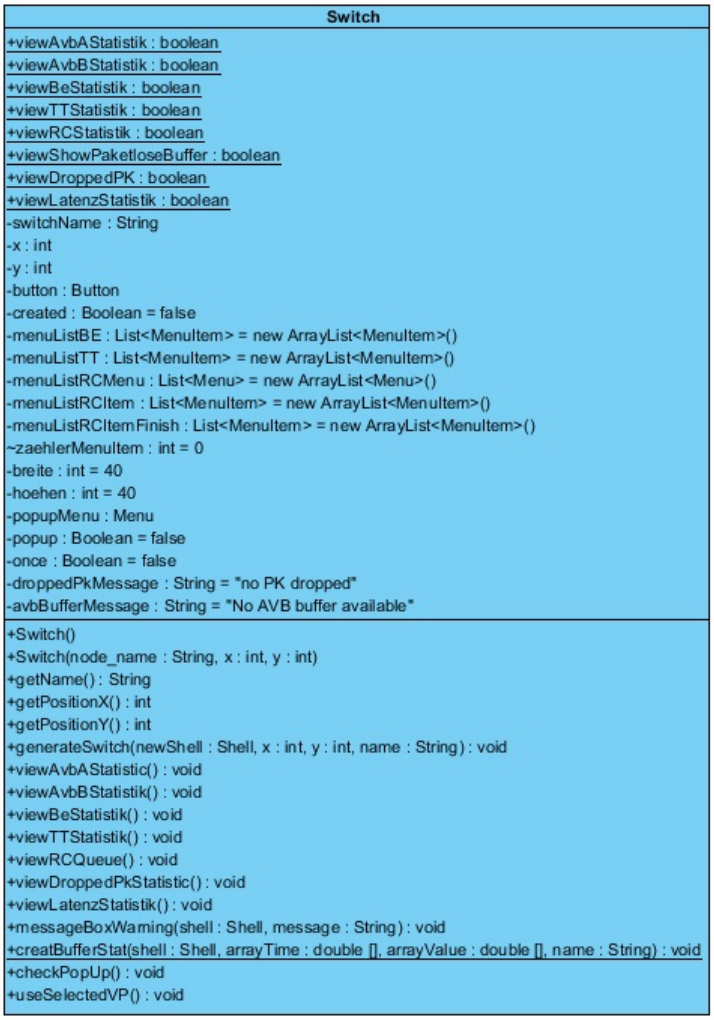


Abbildung 5.7.: Klassendiagram - Switch.java

Connection.java

Abbildung 5.8.: Klassendiagram - Connection.java

Die *Connection.java* Klasse erstellt die Verbindungen zwischen zwei Modulen. Die Informationen, welche Module miteinander verbunden sind, werden der XML-Datei entnommen (siehe Abbildung 5.1). Für die Erstellung einer Verbindung müssen zunächst die *src-Module* und die *dest-Module* aus der XML-Datei abgespeichert werden. Diese abge-

speicherten Module haben x- und y-Koordinaten für die Position, die für die Erstellung der Verbindung benötigt werden. Mit Hilfe der *drawLine* Methode werden diese Verbindungen gezeichnet. Jede Verbindung zwischen zwei Modulen, die in dem OMNeT++ Netzwerk existieren, erzeugt eine *Connection.java* Klasse.

Die Auswahl des Viewpoints bestimmt die Ansichten und Funktionalitäten, die in der *Connection.java*-Klasse zur Verfügung stehen. Diese werden zu Beginn mit der useSelectedVp Methode ausgeführt. Jede der erzeugten *Connection.java* Klassen besitzen ihre eigenen MouseListener, wie z.B die mouseDown Methoden, die von der SWT-Bibliothek zur Verfügung gestellt werden, um auf einen Mausklick reagieren zu können. Mit einem Rechtsklick auf eine Verbindung kann der Nutzer die Linkauslastung oder die Bandbreite über ein Auswahlmenü auswählen, das zuvor mithilfe der MouseListener Methode getriggert wurde. Um sicher zu stellen, dass der Mausklick auch wirklich auf die Verbindung zwischen den zwei Modulen erfolgte, wird anhand der Source- und der Destination-Koordinaten der Module eine Geradengleichung errechnet. Wenn der Mausklick auf der Geraden der jeweiligen *Connection.java*-Klasse war, wird der MouseListener getriggert.

Die Methode *handleMetrikItem* erstellt für die Metriken jeweils einen Listener, der bei der Auswahl des Menüs auf der Verbindung getriggert werden, und bildet den Wert der Linkauslastung oder der Bandbreite des zuvor abgespeicherten Werts aus der Scalar-Datei auf einem Label ab.

5.2.5. Plug-In Konfiguration

Die Konfiguration von Plug-Ins wird mithilfe von zwei Dateien durchgeführt. Diese sind die MANIFEST.MF und plugin.xml Dateien, mit denen beschrieben wird, wie das Plug-In eingebunden wird, welche Extension und Extensionpoints benötigt werden und wie diese gestartet werden. Im Topology Analyzer werden vier Extensionpoints für die Konfiguration verwendet. Im Folgenden werden Auszüge der Extensionpoins als Listing abgebildet und genauer beschrieben. Zum Schluss wird ebenfalls ein Auszug des MANIFEST.MF abgebildet und erläutert [vgl. Daum, 2007, S. 26-27] .

org.eclipse.ui.commands:
Über diesen Extensionpoint können Befehle zugeordnet werden. Dies wird genutzt, um dem Button, welcher sich unter dem Menu befindet (siehe Listing 5.7), einen Namen zuzuordnen. (siehe Listing 5.6 Zeile 9). An diesem Button wird dann der Handler angebunden (siehe Listing 5.8), über den das Programm gestartet wird.

```
1  <extension
2       point="org.eclipse.ui.commands">
3     <category
4          name="Sample_Category"
5          id="topology.analyzer.plugIn.commands.category">
6     </category>
7     <command
8
9          name="Start"
10         categoryId="topology.analyzer.plugIn.commands.category"
11         id="topology.analyzer.plugIn.commands.sampleCommand">
12    </command>
13 </extension>
```

Listing 5.6: plugin.xml - Extensionpoint: Command

org.eclipse.ui.menus:
Dieser Extensionpoint wird genutzt, um das Plug-In an der Menübar von OMNeT++ anzubinden. Dies erfolgt über den locationurl (siehe Listing 5.7 Zeile 4), den die Eclipse-Hauptmenübar als Pfad bekommt. In Zeile 6 wird der Name zugewiesen, über den das Plug-In in der Menüleiste wiederzufinden ist. Zum Schluss wird der Button aus dem Command Extension anhand der ID *topology.analyzer.plugIn.commands.sampleCommand* an dem Menü angebunden (siehe Listing 5.7 Zeile 9 bis 13), sodass dieser mit einem Klick erscheint.

```
1  <extension
2       point="org.eclipse.ui.menus">
3     <menuContribution
4          locationURI="menu:org.eclipse.ui.main.menu?after=additions">
5       <menu
6            label="Topology_Analyzer"
7            mnemonic="M"
```

```
8           id="topology.analyzer.plugIn.menus.sampleMenu">
9         <command
10            commandId="topology.analyzer.plugIn.commands.sampleCommand"
11            mnemonic="S"
12            id="topology.analyzer.plugIn.menus.sampleCommand">
13        </command>
14      </menu>
15    </menuContribution>
16 </extension>
```

Listing 5.7: plugin.xml - Extensionpoint: Menü

org.eclipse.ui.handlers:
Über diesen Extensionpoint wird der Startpunkt des Plug-ins festgelegt. In diesem Fall soll die StartShell Klasse als Einstiegspunkt genutzt werden. Die execute() Methode der Klasse wird dadurch automatisch aufgerufen. Anhand der ID *topology.analyzer.plugIn.commands.sampleCommand* wird dieser an den Command-Button angebunden.

```
1 <extension
2       point="org.eclipse.ui.handlers">
3    <handler
4          commandId="topology.analyzer.plugIn.commands.sampleCommand"
5          class="shell.StartShell">
6    </handler>
7 </extension>
```

Listing 5.8: plugin.xml - Extensionpoint: Handler

org.eclipse.ui.bindings:
Über diesen optionalen Extensionpoint wird eine Tastenkombinationen zum Öffnen des Plug-In hinzugefügt. In Zeile 6 wird die Tastenkombination definiert, über die der Command extensionpoint getriggert und der Handler extensionpoint gestartet wird. Diese zusätzliche Funktion soll dem Nutzer die Möglichkeit geben, das Plug-In komfortabel zu starten.

```
1 <extension
2       point="org.eclipse.ui.bindings">
3    <key
```

```
4         commandId="topology.analyzer.plugIn.commands.sampleCommand"
5         contextId="org.eclipse.ui.contexts.window"
6         sequence="M1+6"
7         schemeId="org.eclipse.ui.defaultAcceleratorConfiguration">
8      </key>
9  </extension>
```

Listing 5.9: plugin.xml - Extensionpoint: Binding

Manifest-Datei:
Die Manifest-Datei enthält Informationen zu den genutzten Bibliotheken, der Versionsnummer, der benötigten Java-Version und dem Namen des Plug-Ins. In Abbildung 5.9 ist das MANIFEST.MF abgebildet. In Zeile 7 werden die benötigten Bibliotheken beschrieben, unter denen sich auch der swt-xy-Graph befindet. Aus Kompatibilitätsgründen wurden die Bundle-Versionen der Bibliotheken entfernt, da OMNeT++ nicht die aktuellsten Versionen der Bibliotheken nutzt. Dies kann dazu führen, dass das Plug-In beim Start der OMNeT++ Umgebung nicht aktiviert wird.

```
1 Manifest-Version: 1.0
2 Bundle-ManifestVersion: 2
3 Bundle-Name: Topology Analyzer
4 Bundle-SymbolicName: topology.analyzer.plugIn;singleton:=true
5 Bundle-Version: 1.0.0.qualifier
6 Bundle-RequiredExecutionEnvironment: JavaSE-1.8
7 Require-Bundle: org.eclipse.ui,
8  org.eclipse.draw2d,
9  xygraph;bundle-version="1.0.0"
10 Export-Package: images,
11  model,
12  shell,
13  viewpoint
```

Abbildung 5.9.: MANIFEST.MF

6. Evaluierung

In diesem Kapitel wird die Evaluierung des Topology Analyzers ermittelt. Zunächst müssen die Evaluierungskriterien festgelegt werden, nach denen das Plug-In evaluiert werden soll.

6.1. Evaluierungskriterien

Die Simulationsergebnisse und die Topologie-Informationen werden von der OMNeT++ Umgebung geliefert. Somit muss lediglich geprüft werden, ob das Plug-In auf denen beim Test ausgewählten Betriebssystemen gestartet werden kann. Des Weiteren müssen die Funktionen anhand der vorhandenen OMNeT++ Projekten getestet werden. Für den Plug-In Test sind folgende Kriterien relevant:

- Konnte das Plug-In erfolgreich gestartet werden?
- Konnten die Topologie-Informationen und die Simulationsergebnisse eingelesen werden?
- Wurde die OMNeT++ Netzwerktopologie auf dem Plug-In richtig abgebildet?
- Sind die Simulationsergebnisse von der Scalar-Datei abgebildet?
- Sind die Statistiken erfolgreich erstellt worden?
- Sind die Ergebnisse von der Scalar- und der Vector-Datei im Topology Analyzer wiederzufinden?

Anhand dieser Kriterien soll entschieden werden, ob das Ziel erreicht wurde und ob das Plug-In die gesetzten Anforderungen erfüllt hat.

6.2. Betriebssystem-Test

Die OMNeT++ Umgebung ist für verschiedene Betriebssysteme verfügbar, darunter Linux, Mac/OS und Windows. Es soll geprüft werden, ob sich das Plug-In auf diesen Betriebssystemen durch das Hinzufügen des *Dropins* Ordner aktivieren lässt. Für das Plug-In ist Java 1.8 erforderlich auf OMNeT++ 4.6 ist Java 1.7 Standard. Um das Plug-In verwenden zu können, muss zunächst OMNeT++ auf Java 1.8 umgestellt werden.

Betriebssystem	OMNeT++ 4.6
Windows:	Funktionsfähig
Linux:	Funktionsfähig
Mac/OS:	Funktionsfähig

Tabelle 6.1.: Betriebssystem-Test des Plug-Ins

Tabelle 6.1 zeigt, dass das Plug-In auf allen Betriebssystemen erfolgreich hinzugefügt und aktiviert werden konnte. Somit ist sichergestellt, dass das Plug-In auf den von OMNeT++ unterstützten Betriebssystemen verwendet werden kann.

6.3. Plug-In

Der Topologie Analyzer wurde auf OMNeT++ 4.6 win64 getestet. Für den Test des Plug-Ins wurden zwei vorhandene Projekte verwendet [CoRE Research Group, b]:

- avb_AS6802
  - small_network
  - large_network

Der Test mit den OMNeT++ Projekten lief ordnungsgemäß ab und alle Topologie- und Metrik-Informationen konnten richtig abgebildet werden. Die Statistiken und Werte im Topology Analyzer stimmten mit den Werten der Scalar- und Vector-Datei überein. Tabelle 6.2 zeigt die getesteten Kriterien.

Evaluierungskriterien	small_network	large_network
Konnte das Plug-In erfolgreich gestartet werden?	Ja	Ja
Konnten die Topologie-Informationen und die Simulationsergebnisse eingelesen werden?	Ja	Ja
Wurde die OMNeT++ Netzwerktopologie auf dem Plug-In richtig abgebildet?	Ja	Ja
Sind die Simulationsergebnisse von der Scalar-Datei abgebildet?	Ja	Ja
Sind die Statistiken erfolgreich erstellt worden?	Ja	Ja
Sind die Ergebnisse von der Scalar- und der Vector-Datei im Topology Analyzer wiederzufinden?	Ja	Ja

Tabelle 6.2.: Evaluierung des Plug-Ins

7. Zusammenfassung/Fazit

Ziel dieser Arbeit war es, die Netzwerktopologie und die auftretenden Metriken eines Echtzeitnetzwerkes mit Hilfe des Plug-In Topology Analyzer visuell darzustellen. Es hat sich herausgestellt, dass die Visualisierung eines OMNeT++ Netzwerkes im Topology Analyzer nicht nur Informationen über die Metriken des Netzwerkes liefert, sondern auch mögliche Fehlerursachen aufweisen kann.

Zu Beginn der Arbeit wurden die Grundlagen dargestellt, um eine einheitliche Wissensbasis zu vermitteln. Der Topology Analyzer verwendet Simulationsergebnisse und Netzwerk-Informationen aus der OMNeT++ Umgebung, somit wurde zu Beginn der Grundlagen die OMNeT++ Umgebung vorgestellt. Danach wurden die Metriken vorgestellt, die in einem Echtzeitnetzwerk auftreten können. Die OMNeT++ Umgebung ist in das Eclipse-Framework eingebettet. Um eine Erweiterung in der vorhandenen Plattform zu ermöglichen, sind Eclipse Plug-Ins am besten geeignet, daher wurden am Ende der Grundlagen die Eclipse Plug-Ins vorgestellt und deren Aufbau und Eigenschaften zum Verständnis für die Implementierung beschrieben.

Im Hauptteil der Arbeit wurden zunächst die Anforderungen an den Topology Analyzer vorgestellt. Anhand der an die verschiedenen Ebenen des Programmes gestellten Anforderungen wurde das Konzept dieser Arbeit erstellt. In diesem Konzept wurde ermittelt, welche Informationen aus der OMNeT++ Umgebung verwendet werden können. Des Weiteren wurde das GUI-Konzept erläutert und die Semantik der Darstellung festgelegt. Daraufhin wurde der Topology Analyzers implementiert. Zum Schluss wurde der Topology Analyzer evaluiert und es wurde festgestellt, dass das Ziel dieser Arbeit erreicht wurde. Der Nutzer ist über den entwickelten Plug-In in der Lage verschiedene OMNeT++ Netzwerke einzulesen und die Simulationsergebnisse visuell darzustellen. Anhand der visualisierten Topologie können verschiedene Netzwerk-Metriken näher betrachtet und analysiert werden. Es konnte festgestellt werden, dass anhand der Daten aus der

Scalar-Datei hohe Latenzen eingegrenzt und anhand der Statistiken näher betrachtet werden können. Durch die Erstellung eines Eclipse Plug-ins konnte das Programm für die Analyse von Netzwerken in die Simulationsumgebung integriert werden und ist somit leicht über die OMNeT++ Umgebung verfügbar.

Der Topology Analyzer ist zunächst nur in der Lage, Komponenten wie z.B. Switches und Nodes darzustellen. In Zukunft könnte dies auf weitere Komponenten erweitert werden, sodass evlt. auch Gateways, CAN-Busse und andere Komponenten visualisiert werden können.

Topology Analyzer Manual

Im Anhang befindet sich das erstellte Manual. Es enthält eine Anleitung zur Bedienung des Programms, eine Installationsanleitung und eine Erläuterung zu den Statistiken und Funktionalitäten des Programmes. Das Manual wurde für den Endnutzer entwickelt.

Topology Analyzer Manual

Burim Mulici
HAW Hamburg
CoRE Research Group

Inhaltsverzeichnis

1 Introduction

Ethernet networks have different kind of metrics. These metrics are analyzed and tested to ensure the operability of the network and the appropriate determine of the cause. The Topology Analyzer is a graphical view of the network and the metrics which are contained. In Figure 1.1 is an example Network with three Nodes and one Switch. Finding the cause of a problem is sometimes annoying, difficult and costs time. The Topology Analyzer gives the opportunity to have an overview of the Metric of the Network.

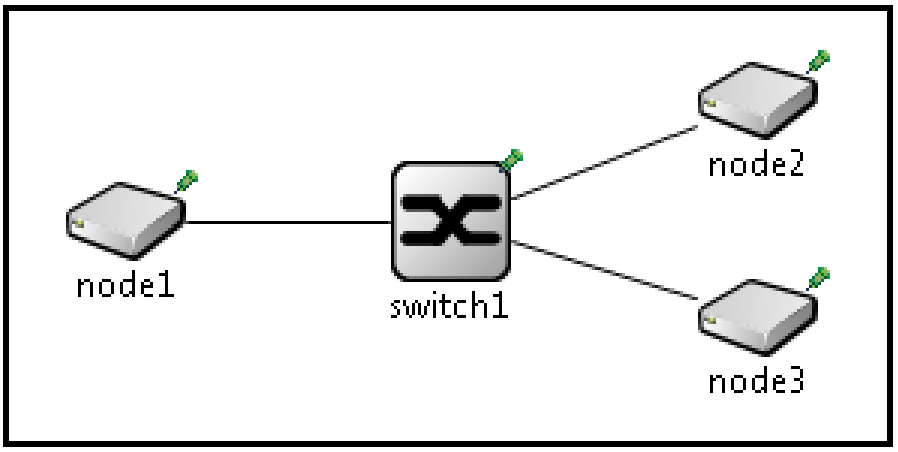


Figure 1.1: Small Network from the CoRE-Examples

2 Getting Started

Before you start using the Topology Analyzer, you need to:

1. Export the Network you want to analyze from OMNeT++ after the network ran for a certain time on the OMNeT ++ simulation.
2. Collect the metric data of the finished Simulation

The following pages will explain how to export the Topology and how to collect the metric Data out of the OMNeT++ Simulation.

Attention: The Plug-In was created and tested under OMNeT++ Version 4.6. Higher Version of OMNeT++ might work, but is not guarranted.

2.1 Exporting the Topology information

Attention:
The Topology Analyzer can only visualize the Module Types Node and Switch at the Moment. More Modules can be implemented in the future.

To make sure the Topology Analyzer will work you have to check two Module Names of the network you want to analyze:

1. The name of the Module Node has to contain the string „node“ (upper and lower case doesn’t matter)
2. the name of the Module Switch has to contain the string „switch“ (upper and lower case doesn’t matter).

The following steps will explain you, how to export the Topology Informations:

Step 1: right click on the Project Explorer > Export

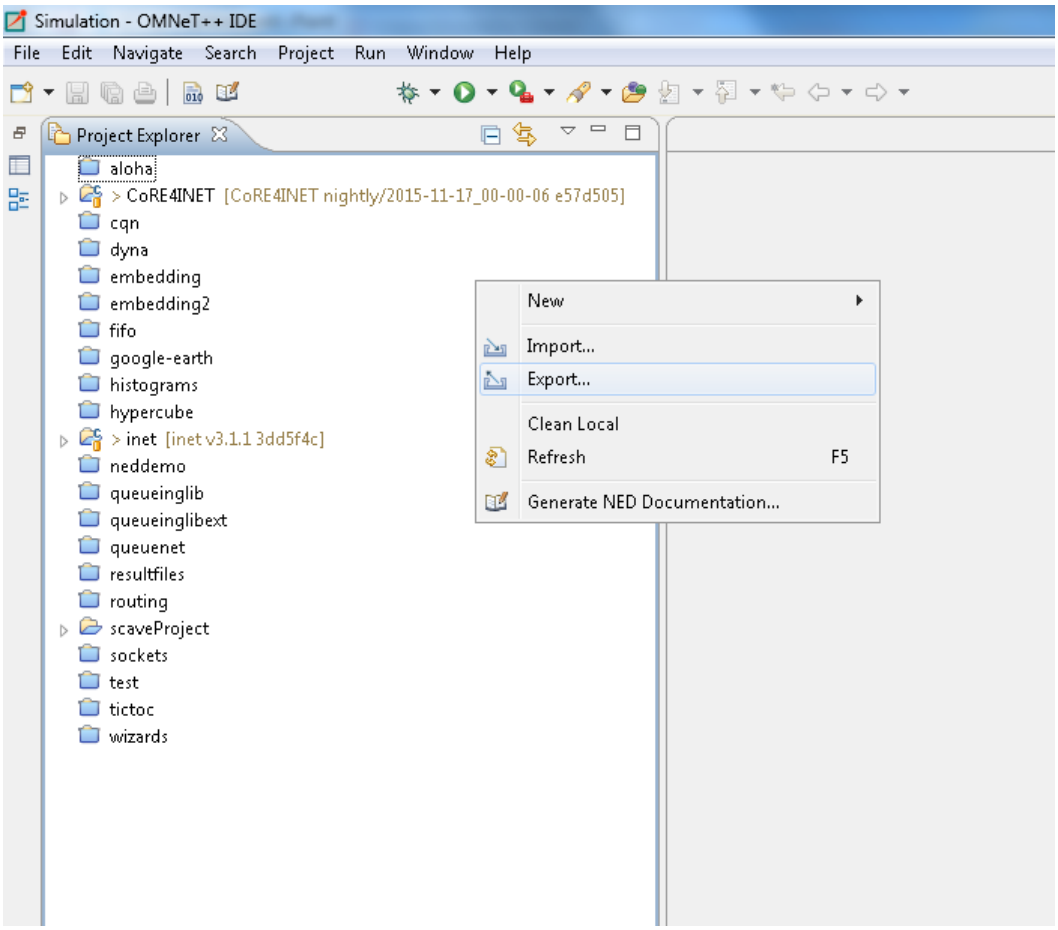


Figure 2.1: OMNeT++ IDE

Step 2: Select OMNeT++/Networks, simulations and Other Items and click the next button

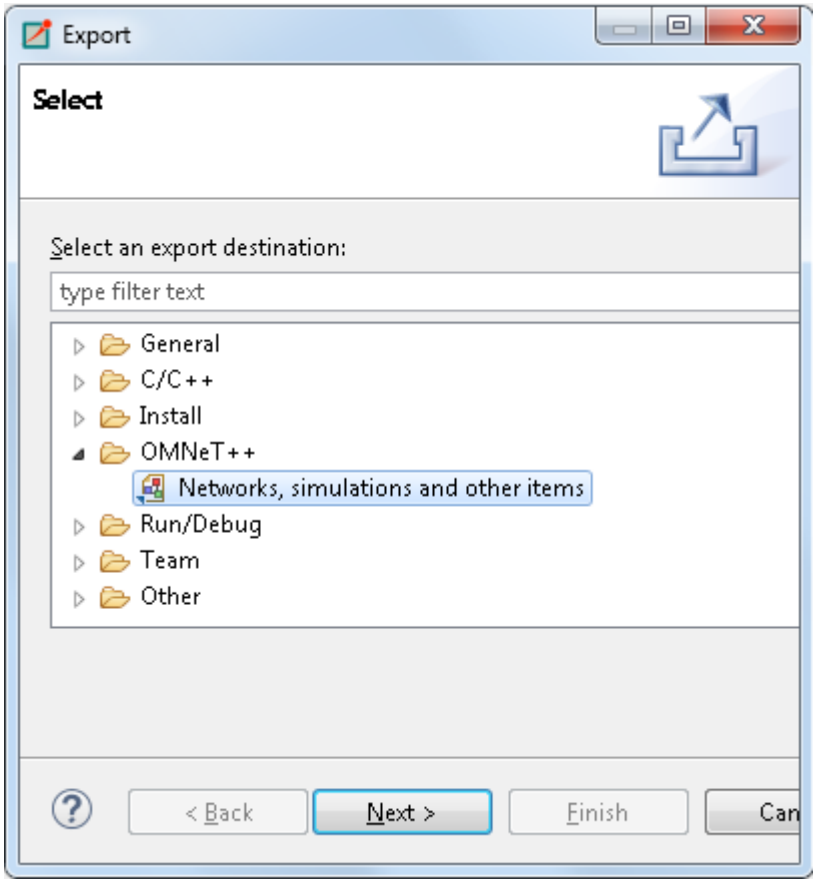
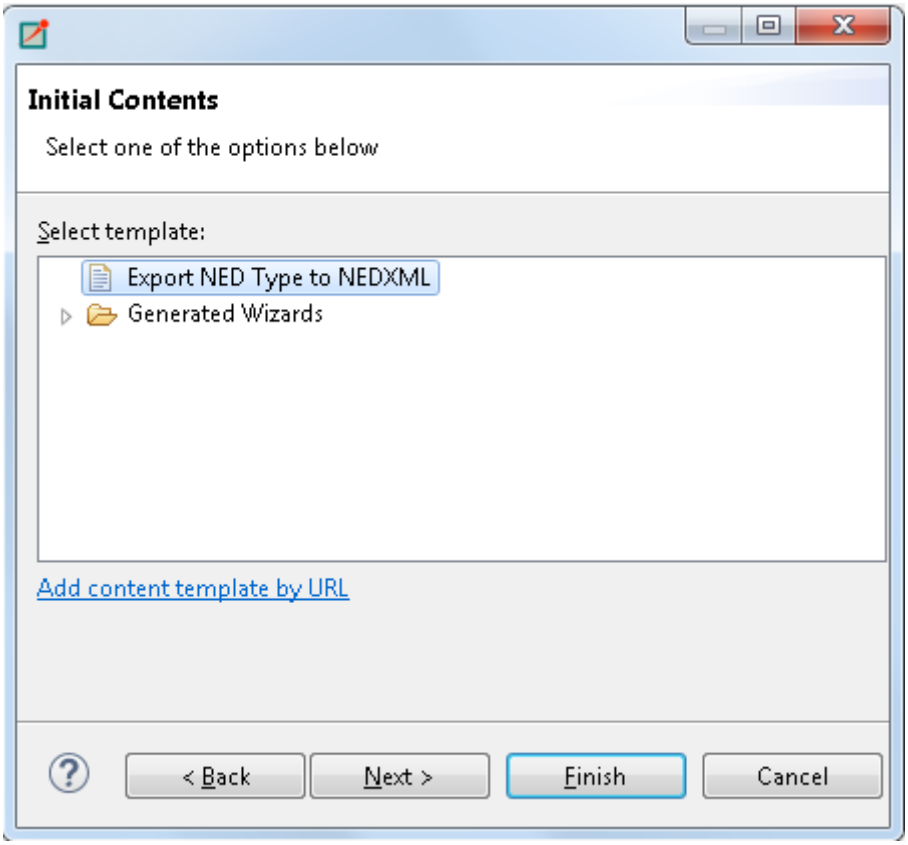


Figure 2.2: Export View

Step 3: Select Export NED Type to NEDXML and click the Finish button



Step 4: Select the Network which should be converted into XML, select the exporting path (for example Figure 2.3) and click Finish

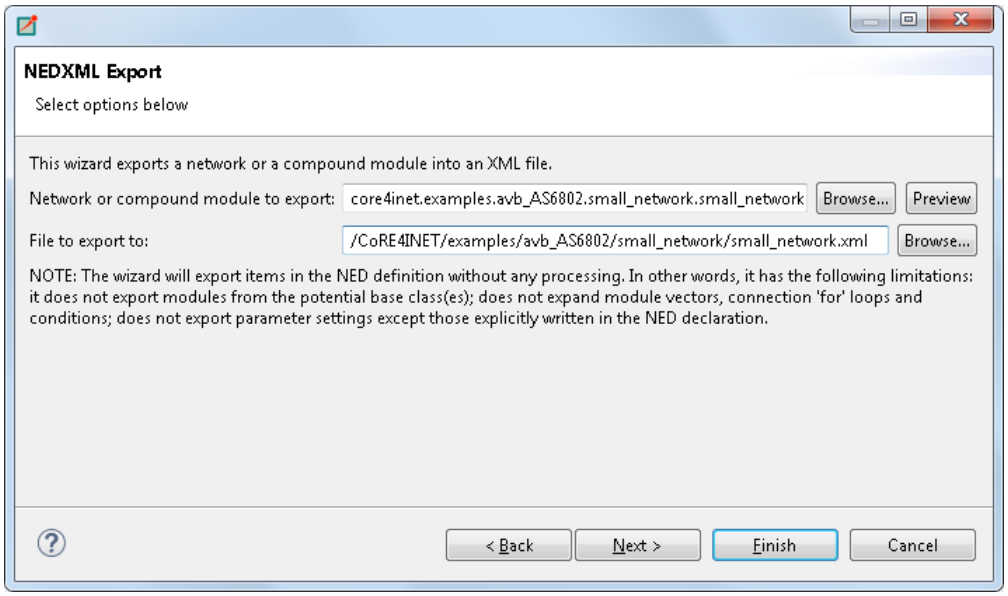


Figure 2.3: XML export example of the small Network

2.2 Collecting the Scalar/Vector Data from your Network

The Metrics are saved in the Scalar and Vector File of the OMNeT++ Simulation. To Start the Plug-in you have to collect the OMNeT++ Simulation data first:

Step 1: Start the simulation and make sure the record is on. The runtime is determined by the user and can be finished anytime.

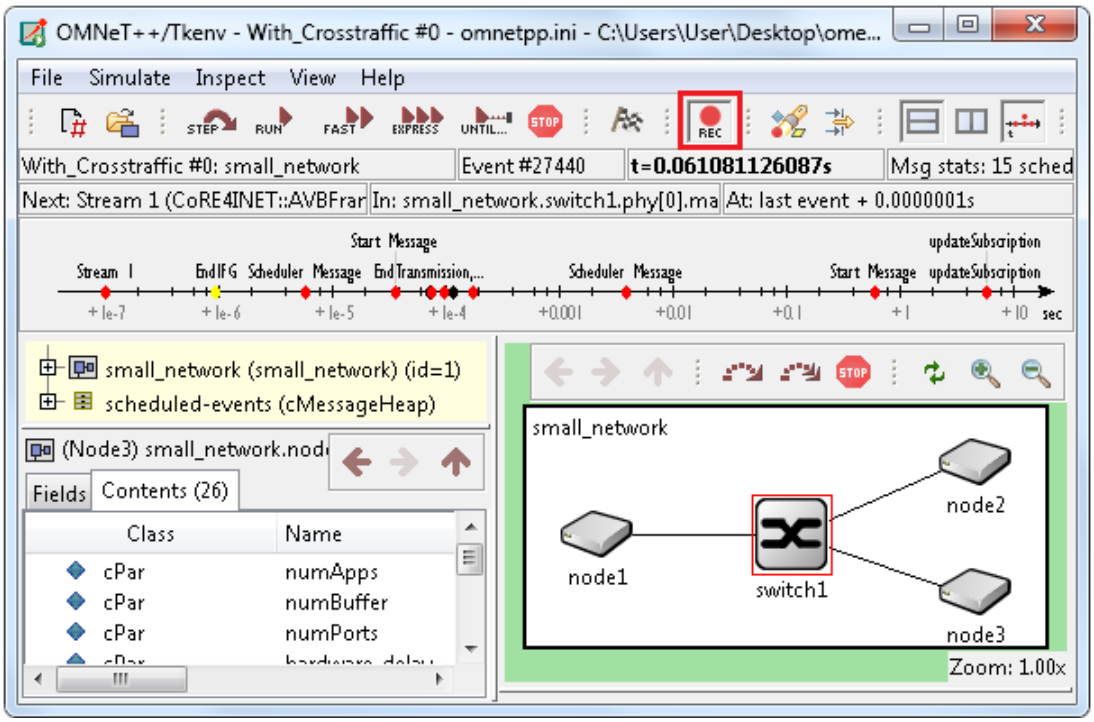


Figure 2.4: OMNeT++ Simulation

Step 2: after the Simulation is finished - go to the result directory and check if the Scalar/Vector File (*.vec and *.sca File) are created (Figure 2.5).

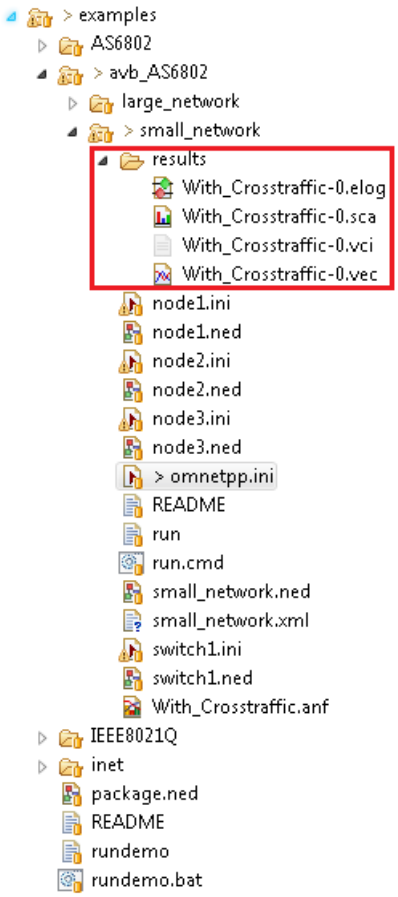


Figure 2.5: Overview of the small network folder

3 Installation

To be able to use this Plug-In Application, the required Java Version under Eclipse must be 1.8 or higher. Under Help -> About OMNeT++ -> IDE -> Installation Detail -> Configuration you can check your current Java Version.

Example:
eclipse.vm=C:/Program Files (x86)/Java/jre1.8.0_66/bin/javaw.exe

If the Java Version is not the required one, then you can change the VM in the omnetpp.ini File under <your Path>OMNeTpp-„version“/ide/omnetpp.ini by adding the following lines under „OMNeT++ IDE“. For example:

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20140415-2008.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.200.v20140603-1326
-name
OMNeT++ IDE
-vm
C:/Program Files (x86)/Java/jre1.8.0_66/bin/javaw.exe
--launcher.XXMaxPermSize
320m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vmargs
-Dosgi.requiredJavaVersion=1.7
-Xms256m
-Xmx1024m
-XX:MaxPermSize=320m
```

Listing 3.1: omnetpp.ini

Example For Windows:

-vm
C:\Program Files(x86)\Java\jre1.8.0_66\bin\javaw.exe **(This is an example Path and can be different!)**

Example for Linux:

-vm
/opt/sun-jre-1.8.0_66/bin/java **(This is an example Path and can be different!)**

Adding the Plug-In Into the OMNeT env.
Copy the Jar-Files out of the TopologyAnalyzer/PlugIn folder to your OMNeTpp directory path at:
<your Path>OMNeTpp-„version“/IDE/dropins
Now restart your OMNeTpp and the Plug-In will be activated automaticlly.

If everything is done correctly, then in the Menubar you can see the Topology Analyzer:

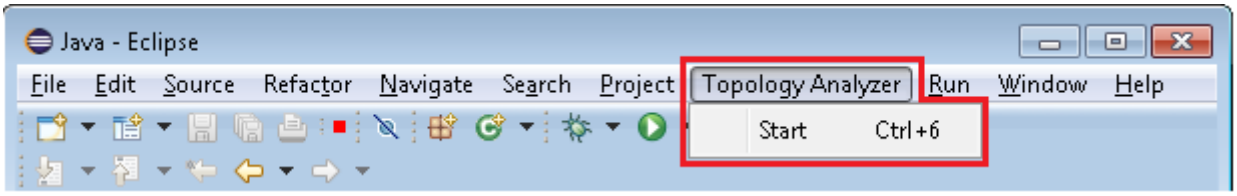
Abbildung 3.1: Menubar View from OMNeT++ 4.6

4 Using the Tool

This section explains the usage of the Topology Analyzer and shows an overview of the funktions of the tool.

4.1 Starting the tool

After the installing of the tool, the user can find the Tool on the menubar of the OMNeT++/Eclipse IDE.



Viewpoint window:
After clicking on start, the first window of the Programm will show up and the User is able to select a Viewpoint. The Viewpoints have the purpois of selecting a certain View for a different useage of the Programm.

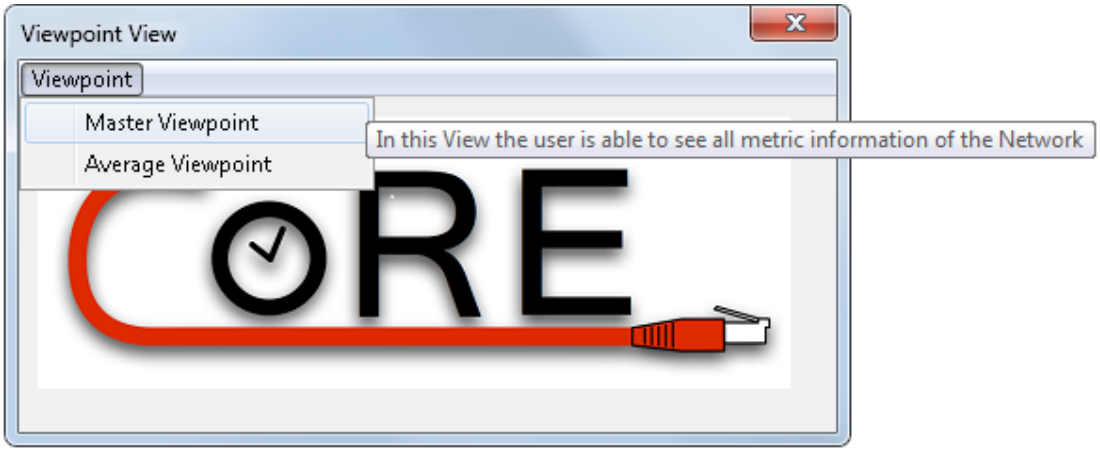


Figure 4.1: Viewpoint View

Currently the Programm has 2 Viewpoints:

- Master View: It contains all the function of the Programm and is able to see all the metric data of the network. Short summary of the shown data:
  - average view of the latencys/jitter of the messages
  - statistic view of the latencys with time/value
  - statistic view of the bufferqueues time/value
  - statistic view of the packet drooped time/value
  - average utilization/idle of a connection
- Average View: It contains only a certain amount of function and is only able to see the average metric data of the network. Short summary of the shown data:
  - average view of the latency/jitter
  - average utilization/idle of a connection

Data Selection View:
In this view the User has to select the XML-File with Open, then the User has to select the Vector file (*.vec) and Scalar file (*.sca).

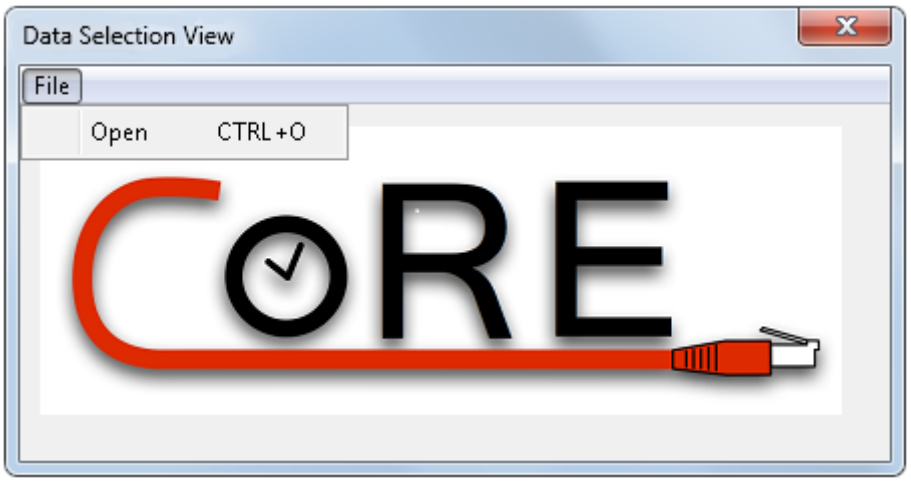


Figure 4.2: View for selecting the files

In this Window the user has to select the topology XML-file of the network.

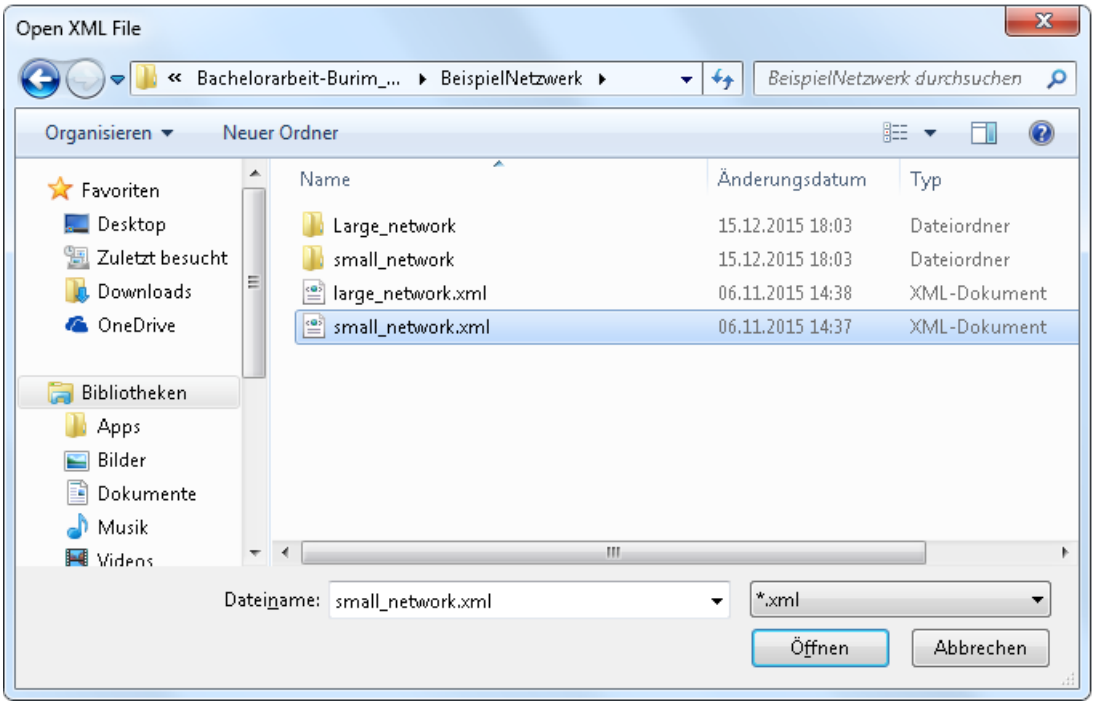


Figure 4.3: Selecting the XML file

In this window the user has to select the Vector file of the network. Please make sure it is the Vector file of the same network of the XML file or the data will be wrong and no error will be throw.

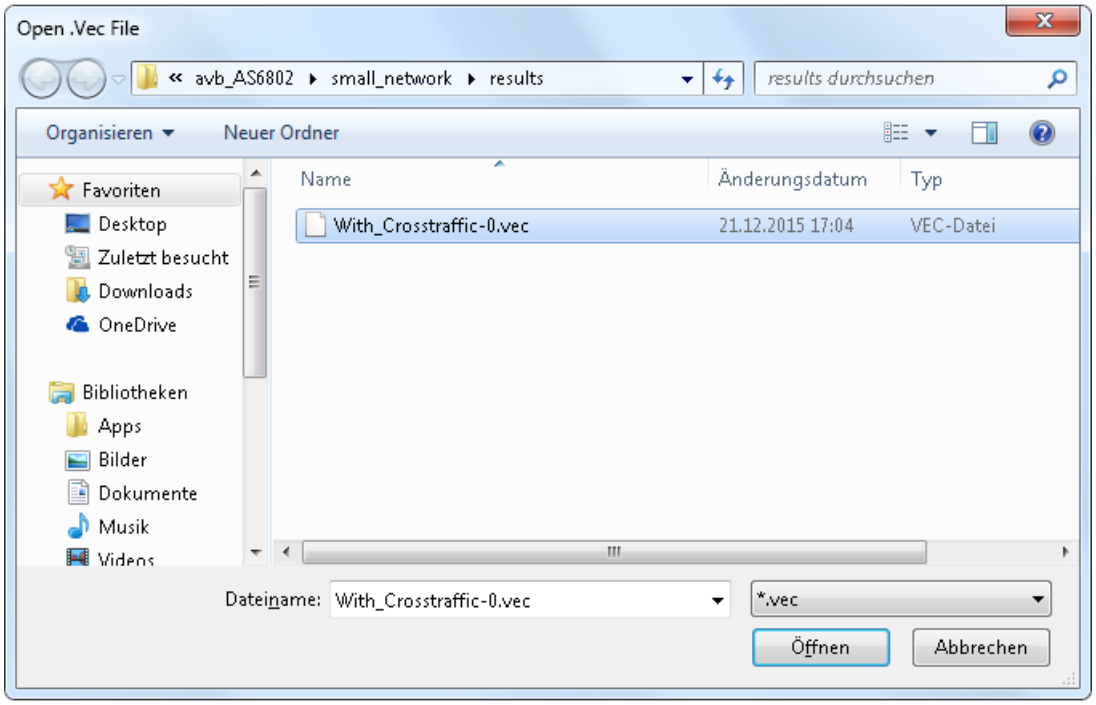


Figure 4.4: Selecting the Vector file

In this window the user has to select the Scalar file of the network. Please make sure it is the Scalar file of the same network of the XML file or the data will be wrong and no error will be throw.

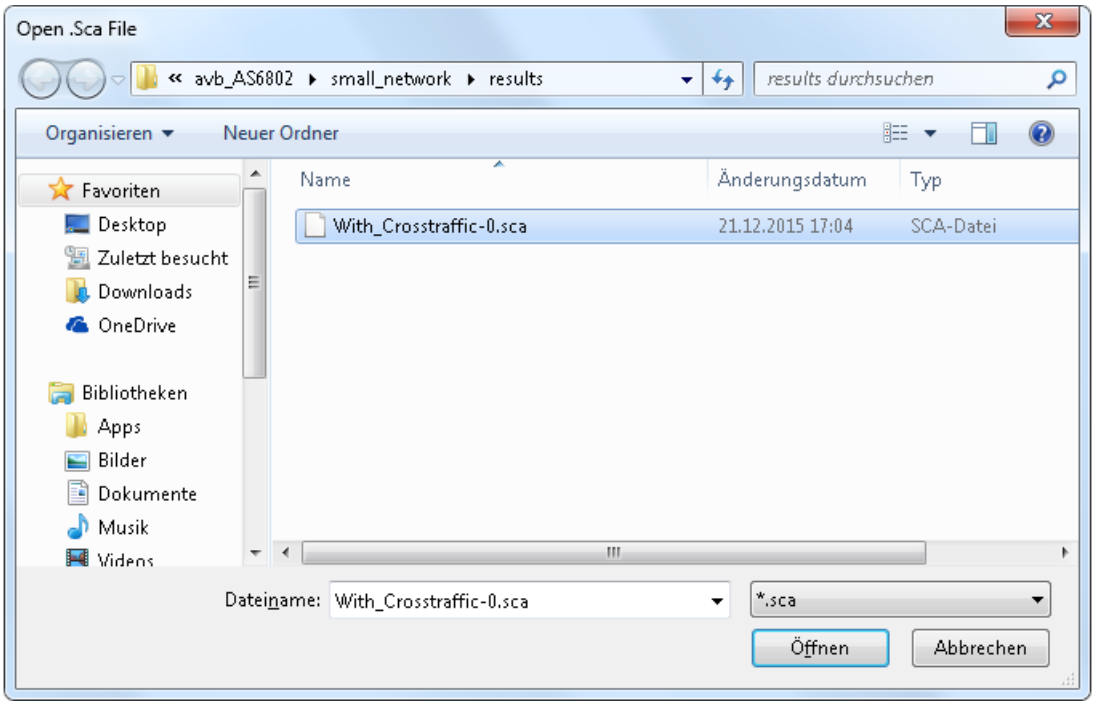


Figure 4.5: Selecting the Scalar file

Topology View:
After selecting the XML, Vector and Scalar file, the topology view of the network will open.

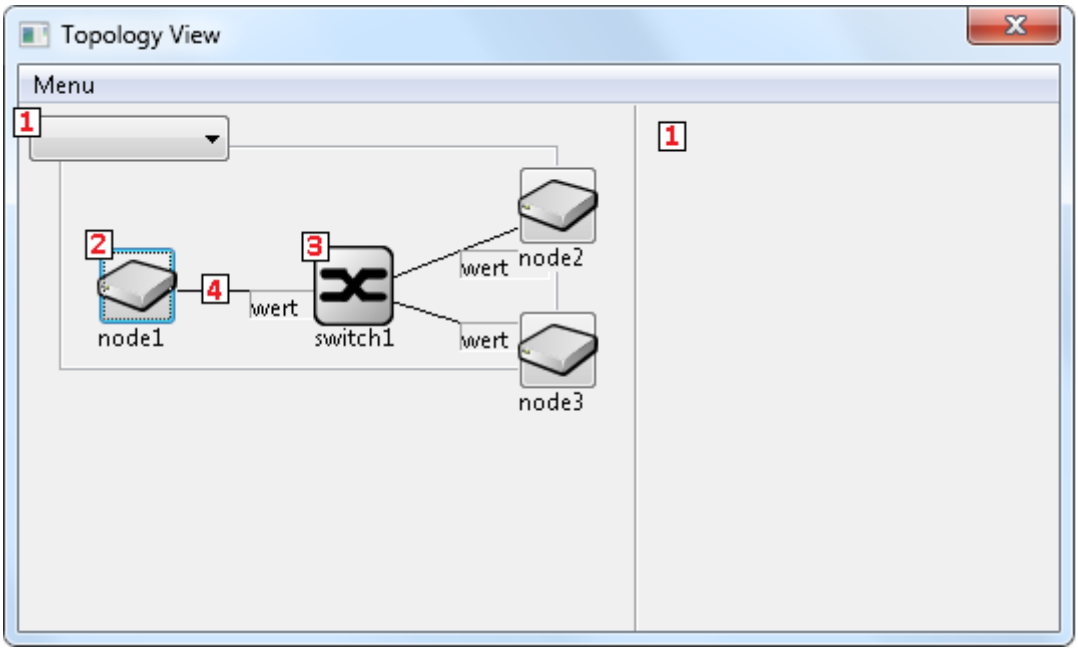


Figure 4.6: Topology View

Quick overview of the functions:

1. Combo box for selecting the average latency/jitter data of the network
2. right click on the node to look for:
   - Buffer queue statistics (avbA/B, TT, RC, BE)
   - Pakage dropped statistic of the node
   - Latency statistics of the differnt kinds of messages (avbA/B, TT, RC, BE)

3. right click on the Switch to look for:
    - Buffer queue statistics (avbA/B, TT, RC, BE)
    - Pakage dropped statistic of the switch
    - Latency statistics of the differnt kind of messages (avbA/B, tt, RC, BE)
4. right click on the connection to look for:
    - Idle value of the connection
    - Utilization value of the connection

Statistic View:
The user is able to anlyze a chosen metric in detail with time/value in this window. The Statistic View is created with the XYGraph plug-in and being used in this application and has functions for simplifying the analyzing. For more Information about XYGraph please visit https://code.google.com/p/swt-xy-graph/

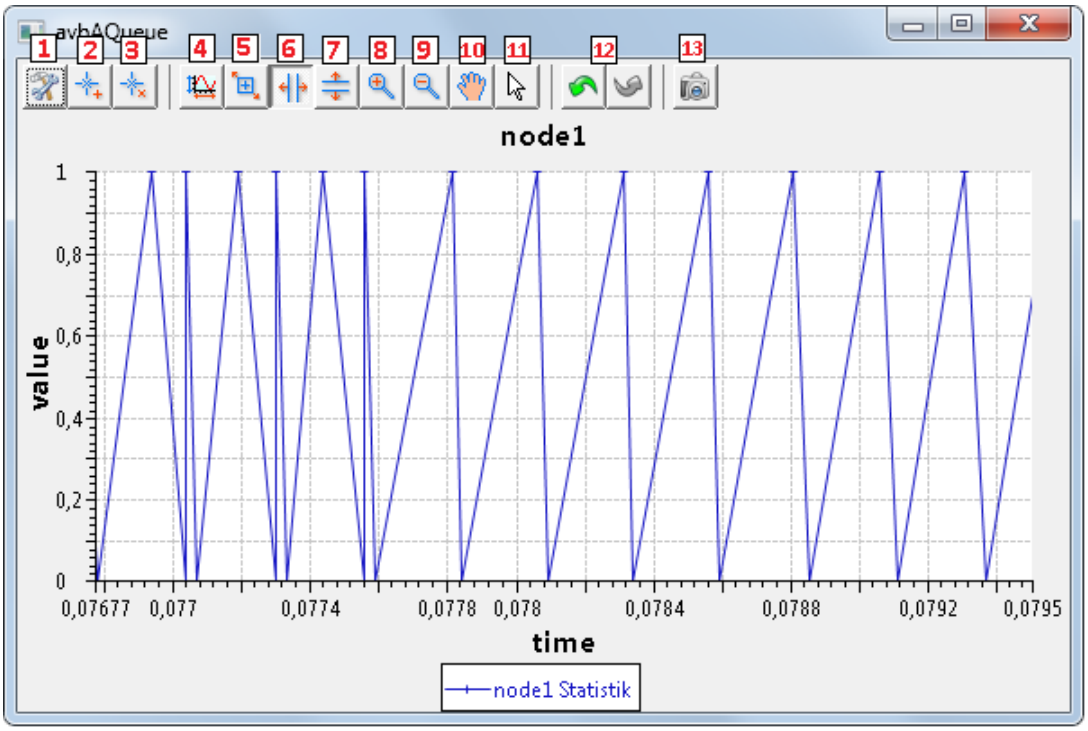


Abbildung 4.7: Statistic View - avbAQueue

Quick overview of the functions:

1. Configuration
2. add Annonation
3. remove Annonation

4. Autoset
5. Rubberband Zoom
6. Horizontal Zoom
7. Vertical Zoom
8. Zoom In
9. Zoom Out
10. Panning
11. None
12. Undo & redo
13. Snapshot function

Literaturverzeichnis

[CoRE Research Group a] CoRE Research Group. – http://core.informatik.haw-hamburg.de/en/ - Zugriffsdatum: 2016-1-22

[CoRE Research Group b] CoRE Research Group: *CoRE4INET Framework*. – URL http://core4inet.realmv6.org/trac/. – Zugriffsdatum: 2015-10-22

[Customer Network Management a] Customer Network Management. – URL https://www.cnm.dfn.de/. – Zugriffsdatum: 2016-01-20. – User Manual des CNM

[Customer Network Management b] Customer Network Management. – URL https://www.cnm.dfn.de/anwendung/. – Zugriffsdatum: 2016-01-20. – User Manual des CNM

[Daum 2007] Daum, Berthold: *Rich-client-Entwicklung mit Eclipse 3.2 - Anwendungen entwickeln mit der Rich Client Platform*. 2. Aufl. Dpunkt-Verlag, 2007. – ISBN 978-3-898-64427-3

[Daum, Berthold 2008] Daum, Berthold: *Rich-Client-Entwicklung mit Eclipse 3.3*. dpunkt.verlag GmbH, 2008. – ISBN 978-3-89864-503-4

[David Megginson] David Megginson. – URL http://www.saxproject.org/about.html. – Zugriffsdatum: 2016-01-25. – SAX is created by David Megginson, Megginson Technologies Ltd. SAX is an Open Source Project and can be used freely. 1998-05-11

[Eclipse Documentation] Eclipse Documentation: Eclipse Foundation. – URL http://help.eclipse.org/mars/topic/org.eclipse.platform.doc.isv/guide/arch.htm?cp=2_0_1 Zugriffsdatum: 2015-12-08

[Eclipse Foundation] Eclipse Foundation: *Eclipse IDE*. – URL http://www.eclipse.org/. – Zugriffsdatum: 2015-12-10

[Eclipse Nebula Visualization] Eclipse Nebula Visualization. – URL http://www.eclipse.org/nebula/widgets/visualization/visualization.php. – Zugriffsdatum: 2016-01-10. – These widgets are originally from CSS BOY project. http://sourceforge.net/projects/cs-studio/

[Evans und Filsfils 2007] Evans, John W. ; Filsfils, Clarence: *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2007. – ISBN 0123705495, 9780080488684

[Fabian Kempf 2008] Fabian Kempf: *RECBAR-Bericht: Metriken in Ethernet-basierten Fahrzeugnetzen*. CoRE Research Group. 2008

[OMNeT++ Community a] OMNeT++ Community. – URL http://www.omnetpp.org. – Zugriffsdatum: 2015-10-20. – OMNeT++ IDE

[OMNeT++ Community b] OMNeT++ Community: OMNeT++ Community. – URL https://omnetpp.org/doc/omnetpp/manual/usman.html. – Zugriffsdatum: 2015-10-22. – OMNeT++ Manual

[OMNeT++ Community c] OMNeT++ Community: OMNeT++ Community. – URL https://omnetpp.org/doc/omnetpp/nedxml-api/group__Overview.html. – Zugriffsdatum: 2016-01-12. – OMNeT++ NEDXML Overview

[OMNeT++ TicToc Example] OMNeT++ TicToc Example: OMNeT++ Community. – URL https://omnetpp.org/doc/omnetpp/tictoc-tutorial/. – Zugriffsdatum: 2015-10-22. – OMNeT++ Example

[Schäuffele und Zurawka 2006] Schäuffele, Jörg ; Zurawka, Thomas: *Automotive Software Engineering - Grundlagen, Prozesse, Methoden und Werkzeuge effizient einsetzen*. 3. akt. u. verb. Aufl. 2006. Wiesbaden : Vieweg+Teubner Verlag, 2006. – ISBN 978-3-834-80051-0

[Shavor, Sherry u. a. 2004] Shavor, Sherry ; Fairbrother, Scott ; D'Anjou, Jim ; Kellerman, John ; Kehn, Dan ; McCarthy, Pat: *Eclipse Anwendung und Plug-*

Ins mit Java entwickeln. PEARSON EDUCATION DEUTSCHLAND, 2004. – ISBN 3-8273-2125-5

[TechTerms] TechTerms: TechTerms. – URL http://techterms.com/definition/tooltip. – Zugriffsdatum: 2016-03-27. – ToolTip Definition

[Varga und Hornig 2008] Varga, András ; Hornig, Rudolf: An overview of the OMNeT++ simulation environment. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, networks and systems & workshops*. New York : ACM-DL, März 2008, S. 60:1–60:10. – URL http://portal.acm.org/citation.cfm?id=1416222.1416290. – ISBN 978-963-9799-20-2

[Vogel und Milinkovich 2015] Vogel, Lars ; Milinkovich, Mike: *Eclipse Rich Client Platform - The Complete Guide to Eclipse Application Development; [revised Edition Based on Eclipse 4.4]*. 3. Aufl. Vogella, 2015. – ISBN 978-3-943-74713-3

Abkürzungsverzeichnis

OMNeT++ Objective Modular Network Testbed 3

NED Network Description 5

QoS Quality of Service 6

SWT Standart Widget Toolkit 12

PDE Plug-In Developer Enviroment 10

IDE Integrated Development Environment 10

JDT Java Development Tooling 10

XML Extensible Markup Language 12

CNM Customer Network Management 29

DFN Deutschen Forschungsnetzes 28

SAX Simple API for XML 45

CoRE Communication over Realtime Ethernet 1

HAW Hochschule für Angewandte Wissenschaften 1

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 3. Mai 2016 Burim Mulici