ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Information Security
www.infsec.ethz.ch

Markus Frauenfelder

Representation of a Network

Semester Thesis
Summer Semester 2005
ETH Zürich, 6. Juni 2005

Supervisor: Diana Senn
Professor: David Basin

Abstract

Eine Firewall schützt ein vertrauenswürdiges Netzwerk vor einem vertrauensunwürdigen Netzwerk. Der Verkehr durch eine Firewall wird auf Grund einer Strategie beobachtet und gefiltert. Um zu verifizieren, ob eine Firewall der Strategie genügt, muss die Firewall getestet werden.

Um solche Tests durchzuführen muss ein Netzwerk beschrieben werden können. Menschen und Computer haben häufig unterschiedliche Ansichten, was eine verständliche Netzwerkbeschreibung ist.

In dieser Semesterarbeit geht es darum, die Netzwerkbeschreibung, die ein Mensch am ehesten versteht (ein Bild) in diejenige, die ein Computer am ehesten versteht (eine Textdatei) umzuwandeln. Um dieses Ziel zu erreichen gilt es, ein Programm zu schreiben. Es handelt sich dabei um einen Parser, der sowohl Textfiles wie auch Graphikfiles einlesen und schreiben kann. Somit wird es möglich, eine Repräsentation eines Netzwerkes für beide involvierten Parteien (Computer und Mensch) einfach lesbar zu machen.

A firewall protects a trusted network from an untrusted network. The traffic is monitored and filtered by the firewall considering a security policy. To verify that the firewall system works as intended tests have to be performed.

In order to run such tests networks have to be described. Humans and computers often have different opinions what a comprehensible description of a network is.

This semester thesis is about converting a network description best understood by humans (a picture) into one best understood by a computer (a plain text file). To achieve this goal a program has to be written. It will be a parser which is capable of parsing and writing plain text files on one hand and pictures on the other hand. Therefore it will be possible to make a representation of a network readable for both parties involved (computers and humans).

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Netzwerke

Es existieren viele verschiedene Möglichkeiten, wie ein Netzwerk repräsentiert werden kann.
Grundsätzlich gibt es zwei verschiedene Möglichkeiten:

- graphische Darstellung
- textuelle Darstellung

Dabei werden die graphischen Darstellungen von Menschen, die textuellen von Computern bevorzugt, beziehungsweise das Textuelle von Menschen und das Graphische von Computern nur sehr schlecht verstanden.

Um Firewalls zu testen, muss man das betroffene Netzwerk kennen. Diese Informationen können auf verschiedene Arten gesammelt werden:

- von jemandem gezeichnet
- mit Text-Dateien technisch beschrieben

Abhängig von der Methode, wie diese Daten erzeugt werden, liegt eine unterschiedliche Form der Spezifikation des Netzwerkes vor. Für die automatische Generierung von Testfällen braucht es aber zwingend eine textuelle Repräsentation. Um in einer späteren Phase die Wege von Netzwerkpaketen visualisieren zu können, was ganz angenehm wäre, braucht es eine Möglichkeit, eine textuelle Repräsentation in eine graphische umzuwandeln und umgekehrt.

2. Tool Evaluation

2.1. Kriterien

Damit die Evaluation nicht zu einem Raten und Wünschen ausartet habe ich im voraus eine Liste von Kriterien aufgestellt, denen eine zu verwendende Grafiksoftware genügen muss:

- Speicherformat: Eine beliebige textuelle Form, die einfach eingelesen und geparst werden kann.
- Einfache Handhabung
- Bemerkungen zu Knoten und deren Verbindungen müssen möglich sein.
- Einfache Repräsentation der Knoten und Verbindungen im gespeicherten oder allenfalls exportierten File.

Einige dieser Kriterien sind besser messbar als andere. Ausserdem haben sich im Laufe der Evaluation weitere Gründe ergeben, weshalb eine betrachtete Software Plus– oder Minuspunkte bekommen kann. Diese Gründe sind aber zum Teil sehr spezifisch für die einzelnen betrachteten Programme. Deshalb macht es wenig Sinn, sie hier aufzuführen.

Ich habe eine längere Liste von Programmen von [3] betrachtet. Es folgt eine Auswahl inklusive der Kriterien, die zu einer Auf– beziehungsweise zu einer Abwertung führten.

2.2. dia

`http://www.gnome.org/projects/dia/`
Bei `dia` habe ich die folgenden positiven Punkte festgestellt:

- Schon viele Shapes vorhanden.
- Bereits einer grösseren Benutzergruppe bekannt.
- (Kann auch für UML verwendet werden).

Wobei der letzte Punkt für diese spezielle Evaluation von geringer bis keiner Bedeutung ist. Die folgenden negativen Punkte fielen mir bei genauerer Ansicht auf:

- Vorhandene Shapes stimmen nicht wirklich richtig mit meinen Vorstellungen überein.
- Verbindungen sind nicht mit Text versehbar.
- Text nicht spezifisch zu Connection-Points möglich beziehungsweise kompliziert. Somit könnten Interfaces nicht direkt angeschrieben werden.
- Das Speicherformat ist zwar in ein XML Format gepresst worden, nützt dessen Vorteile aber nicht aus: anstatt z.B. ein Element mit einem spezifischen Namen (z.B. "<width> 3cm</width>") zu erstellen, wird einfach ein generelles Element kreiert und dann in dem Text zu diesem Element der Name der Eigenschaft erwähnt ("< item>width = 3cm</item>") .

Da `dia` für mich keine Eigenschaften, die unbedingt für einen Gebrauch sprachen, aufwies, hatten die negativen Kriterien am Schluss das stärkere Gewicht (vor allem, dass Interfaces nicht angeschrieben werden können), was zu einer Ablehnung von `dia` führte.

2.3. tgif

`http://bourbon.usc.edu:8001/tgif/index.html`
Auch bei `tgif` waren die positiven Punkte einfach zu finden:

- Als Speicherformat wird Prolog-Syntax verwendet, was das einlesen einfacher machen würde, da eine klar definierte Sprache verwendet wird.
- Bemerkungen / Annotationen zu Shapes sind möglich.
- Zwar sind nur wenige Shapes vorhanden, diese wären aber ausreichend.

Zu diesen vielen positiven Punkten habe ich relativ schnell einen sehr negativen Punkt gefunden: leider ist das Erstellen von Verbindungen, die dann auch als solche erkannt werden, entweder nicht möglich oder aber ausserordentlich schlecht dokumentiert. Auf jeden Fall habe ich nicht gefunden, wie sich solche Verbindungen erstellen lassen. Das hat dann auch zu einer Ablehnung von `tgif` geführt.

2.4. tcm

`http://wwwhome.cs.utwente.nl/~tcm/`
`TCM` oder genauer `tssd` des `TCM` (Toolkit for Conceptual Modelling) macht auf den ersten Anblick einen sehr rudimentären Eindruck. Ich hatte am Anfang das Gefühl, vor einer eher veralteten Sofware zu sitzen. Und zwar vor allem, was das Design derselben angeht.

Trotz dieses schlechten ersten Eindruckes habe ich mich entschieden, `tssd` doch noch genauer zu betrachten. Es haben sich die folgenden positiven Punkte herauskristallisiert:

- Die bereits vorhandenen Shapes haben schon gute, brauchbare Formen und Eigenschaften (Annotationen derselben ist möglich).
- Gute Verbindungen zwischen Shapes möglich (inkl. Annotation derselben).

- Das Speicherformat ist einfach lesbar und klar aufgebaut.
- `TCM` ist OpenSource.
- Sowohl das Fileformat und ganz allgemein die Software sind gut dokumentiert.
- Zwar gewöhnungsbedürftige, dann aber einfache Handhabung.

Allen diesen positiven Punkten stand bloss die doch relativ spezielle Handhabung gegenüber. Die logische Folge war, diese Software für die Semesterarbeit weiter zu verwenden.

2.5. Weitere betrachtete Tools

Neben den oben genauer beschriebenen Tools, habe ich noch weitere Software betrachtet. Sehr häufig hat schon der erste Anblick zu einer raschen Ablehnung geführt. Diese jeweiligen Gründe sind stichwortartig bei der jeweiligen Software aufgeführt.

Xfig Sehr unbequemes Handling der Verbindungen zwischen den Shapes.

Sketch Keine Shapes.

Sodipodi Keine Shapes.

QCad Komisches Fileformat.

Inkscape Shapes sind gemäss Dokumentation zwar vorhanden, gefunden habe ich sie aber nicht.

oCADis Die Homepage ist sehr veraltet, was auf ein Projekt schliessen lässt, das nicht mehr gewartet wird.

Babygimp Ein Icon-Editor.

Guppi Zum plotten von Graphen.

Impress Homepage zum Evaluationszeitpunkt offline.

Gimp Unterstützt kein Speicherformat für Vektorgraphiken.

lvtools Keine Zusammengehörigkeit darstellbar.

Gaphor In erster Linie ein UML-Editor.

DiaCanvas2 Nur eine Library.

Tux Paint Ein Zeichenprogramm für Kinder.

Metagraph-3D Unterstützt weder Vektorgraphiken noch Shapes.

Zusätzlich zu den erwähnten Programmen sind noch einige weitere ausgeschieden, da sie kommerziell sind:

- Rational Rose
- PhotoGenics
- Xebot
- JPhotoBrush
- Robochart

2.6. Entscheidung

Wie weiter oben schon angeführt, haben bei diversen Tools, die ich gefunden habe, relativ schnell negative Punkte ins Auge gestochen. Häufig waren das Punkte, wie das Fileformat, das nicht brauchbar war, die mangelhafte Handhabung oder die mangelhafte / fehlerhafte Dokumentation. Am meisten positive Punkte konnte ich bei `tssd` finden. Ausserdem konnte ich dort bloss den einen negativen Punkt, der etwas gewöhnungsbedürftigen Handhabung feststellen. Zum Schluss war die Wahl von `tssd` einfach: es gab keine besseren Alternativen.

3. Design

Die gesamte zu schreibende Software wird schlussendlich nicht viel anderes machen, als ein Fileformat in ein anderes überführen. Das eine der beiden Formate ist durch die Wahl von `tssd` bereits gegeben. Das andere gilt es im laufe dieser Semesterarbeit zu definieren. Ist diese Definition gemacht, beruht der Rest der Arbeit aus den folgenden Schritten:

- Parsen eines Inputfiles.
- Internalisieren des Inputfiles. Daraus entsteht eine interne Repräsentation (IR).
- Ausgabe im entsprechenden Outputformat.

3.1. Netzwerke

Ein Netzwerk, in welchem eine Firewall klassisch steht, besteht aus folgenden Objekten:

- Knoten
- Verbindungen dieser Knoten
- Beschreibungen von Knoten und deren Verbindungen

Wobei Knoten in diesem Kontext verschiedene Dinge sein können: zum einen werden einfache Computer in dem Netzwerk als Knoten dargestellt, gleichzeitig werden aber auch ganze Subnetze als Knoten dargestellt.

Da ich versucht habe den gesamten Programmcode in Englisch zu halten, werde ich im weiteren Knoten auch als Units und Verbindungen als Connections bezeichnen.

Wenn diese Dinge in ausreichend präzisem Masse beschrieben werden können, kann man daraus ein Netzwerk generieren.

3.2. Name

Um dem zu erstellenden Programm einen Namen zu geben, habe ich eine kleine Umfrage gestartet. Die erste einigermassen sinnvolle und dabei auch noch genügend kurze Antwort war “NetMap”.

3.3. Softwaredesign

Im Kern der zu schreibenden Software steht ein Lexer und ein Parser. Den Lexer habe ich mit flex [2] generieren lassen. Da die Dokumentation, welche ich auf dem Internet über Parsergeneratoren, welche echtes C++ (nicht bloss C mit einem C++ Compiler übersetzt) erzeugten, mir unvollständig und auch unverständlich erschien, habe ich mich entschieden, den Parser, der Einfachheit halber, selber zu schreiben.

Als Folge davon konnte der Parser auch soweit angepasst werden, dass er eine interne Repräsentation des Netzwerkes, gemäss Abbildung 1, enthält.

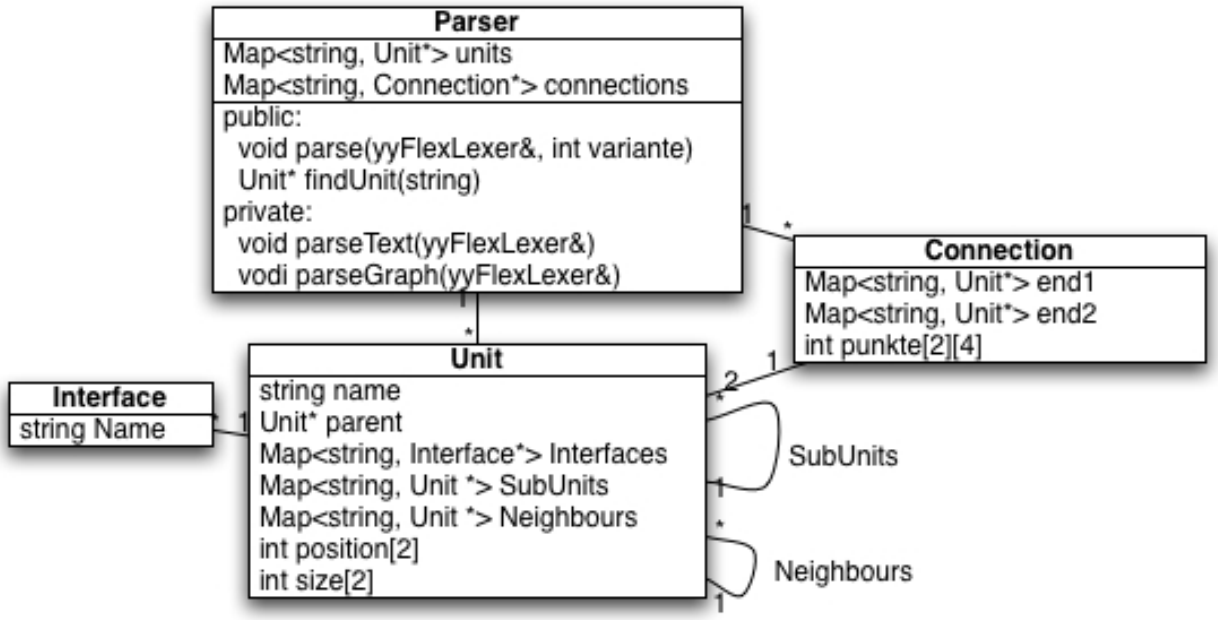

Abbildung 1: Interne Repräsentation des Netzwerkes

Dabei sind folgende Sachverhalte zu betonen:

- Der Parser kennt sowohl eine Map aller Units, wie auch eine Map aller Connections.
- Jedes Unit kennt
  - seine eventuell vorhandenen Subunits.
  - seine Nachbarn.
  - seine Interfaces.
- Jede Connection kennt ihre beiden Enden (Units).

4. Implementation

4.1. Entwicklungsumgebung

NetMap wurde unter Gentoo Linux mit einem 2.6.11 -er Kernel entwickelt. NetMap wurde in C++ implementiert und mit gcc 3.3.5 kompiliert. Ausserdem wird zum Kompilationsprozess make 3.80 verwendet. Für die Generierung des Scanners wird flex 2.5.31 gebraucht.

4.2. Textuelles Dateiformat

Um im Format `mfr` ein Netzwerk zu beschreiben, bedient man sich der folgenden 'Methoden':

Erstellen von Einheiten

`add( was, ID, Name, Service);`
- **was** <Netzwerk, Host, DMZ>Beschreibt die Art des Units, das hinzu gefügt werden soll.
- **ID** eine eindeutige Kennzeichnung des Units.
- **Name** Name des Units (in doublequotes).
- **Service** Beschreibung des Services, welcher dieses Unit erbringen soll.

Erstellen von Sub-Einheiten

`subAdd(was, ID, Name, wozu, Service);`
- **was** <Netzwerk, Host, DMZ>Beschreibt die Art des Units, das hinzu gefügt werden soll.
- **ID** Eine eindeutige Kennzeichnung des SubUnits.
- **Name** Name des SubUnits (in doublequotes).
- **wozu** Zu welchem Unit gehört dieses SubUnit.
- **Service** Beschreibung des Services, welcher dieses SubUnit erbringen soll.

Erstellen von Interfaces

`ifAdd(Einheit, Name, IP(-Range), Eigenschaften, Netz dahinter);`
- **Einheit** Zu welchem Unit dieses Interface gehören soll.
- **Name** Name des Interfaces.
- **Eigenschaften** Eine einfache Beschreibung der Eigenschaften.
- **Netz dahinter** Ein String, welches Netz mit diesem Interface verbunden ist.

Erstellen von Verbindungen zwischen den Einheiten

`connection(ID, EndPunkt1, EndPunkt2, NameInterface1, NameInterface2);`
- **ID** Eine eindeutige Kennzeichnung der Connection.
- **EndPunkt1** ID des Units, welches das eine Ende der Verbindung repräsentiert.
- **EndPunkt2** ID des Units, welches das andere Ende der Verbindung repräsentiert.
- **NameInterface1** Name, des Interfaces beim EndPunkt1, welches mit dieser Verbindung verbunden sein soll.
- **NameInterface2** Name, des Interfaces beim EndPunkt2, welches mit dieser Verbindung verbunden sein soll.

Erstellen von Beschreibungen zu Einheiten / Verbindungen
`annotationAdd(ID, Beschreibung);`
- **ID** Welches Unit / Welche Connection soll diese Annotation erhalten.
- **Beschreibung** Eigentliche Annotation (in doublequotes).

Hier gilt es zudem zu beachten, dass Beschreibungen zu Interfaces direkt bei den Einheiten (Units) erstellt werden. Dies kommt daher, dass ein Interface im Kontext von `NetMap` nichts weiteres ist, als eine Eigenschaft einer Einheit.

4.3. Graphisches Dateiformat

Im Dateiformat von `tssd` werden die Beschreibungen der zu zeichnenden Objekte von deren Positionierung getrennt gespeichert. Konkret bedeutet das, dass zum Beispiel ein Unit zuerst mit dessen Name, Annotation usw. aufs File geschrieben wird, während die Informationen über dessen Platzierung erst in einer späteren Phase folgen. Ein analoges Vorgehen besteht bei den Verbindungen zwischen zwei Units. In einem ersten Schritt erfolgt die Beschreibung der Verbindung mit der Angabe des Namens, der Annotation sowie der beiden Enden (inklusive Namen der dortigen Interfaces). In einem zweiten Schritt, später in der gespeicherten Datei, erfolgt die Beschreibung der Geometrie der Verbindung (der Startpunkte sowie des Verlaufs).

4.4. Positionierung von Units

Um die Units, welche in dem Textfile beschrieben werden, platzieren zu können, musste ich einen Algorithmus finden. Da ich es als nicht eigentliche Hauptaufgabe dieser Semesterarbeit betrachtet habe, **schöne** Netzwerke zu zeichnen und weil eben dieses schöne zeichnen eine sehr komplexe Sache ist, habe ich einen ganz einfachen Algorithmus implementiert:

- Jedes Unit bekommt einmal den Aufruf `position(int alpha, int x, int y)`. Es wird sich selbst an die Position (x,y) setzen und alle seine n Nachbarn auch den Befehl geben, sich zu positionieren.
- Jedes Unit platziert seine Nachbarn so, dass diese im Kreis um es herum angeordnet werden. Dabei wird alle `alpha` Grad ein Unit platziert.
- Die Positionierung von drei aufeinanderfolgenden Units wird nicht als Gerade mit drei Punkten, sondern als zwei Geraden, die in einem Winkel von `alpha` Grad von einer Geraden abweichen, erzeugt.

4.5. Aufbau

NetMap besteht aus den folgenden Klassen:

- Unit
- Connection
- Network : Unit
- Firewall : Unit
- Host : Unit
- Graph
- Text
- Interface
- Parser

- yyLexer (Scanner.cpp, Scanner.h), erstellt durch flex aus mfr.l
- Token

Von allen diesen Klassen sind jeweils `.h` sowie `.cpp` Files vorhanden. Dabei ist die Klasse selbst sowie alle get und set Methoden im .h File beschrieben. Alle weiteren Methoden sind im .cpp File beschrieben.

Ausserdem wurden noch die folgenden Files zur Implementation erstellt:

- `NetMap.cpp` (enthält die main-Methode)
- `misc.h`, `misc.cpp` (ganz kleine Helfermethoden, die sonst niergens rein passen.)

Alle implementierten Methoden können in der jeweiligen Source nachgelesen werden. Ich möchte hier nur auf ein paar Spezialitäten eingehen. Welche Zusammenhänge die einzelnen Klassen miteinander haben, sollte aus Abbildung 2 soweit klar werden, dass die folgenden Erläuterungen verständlich sind.

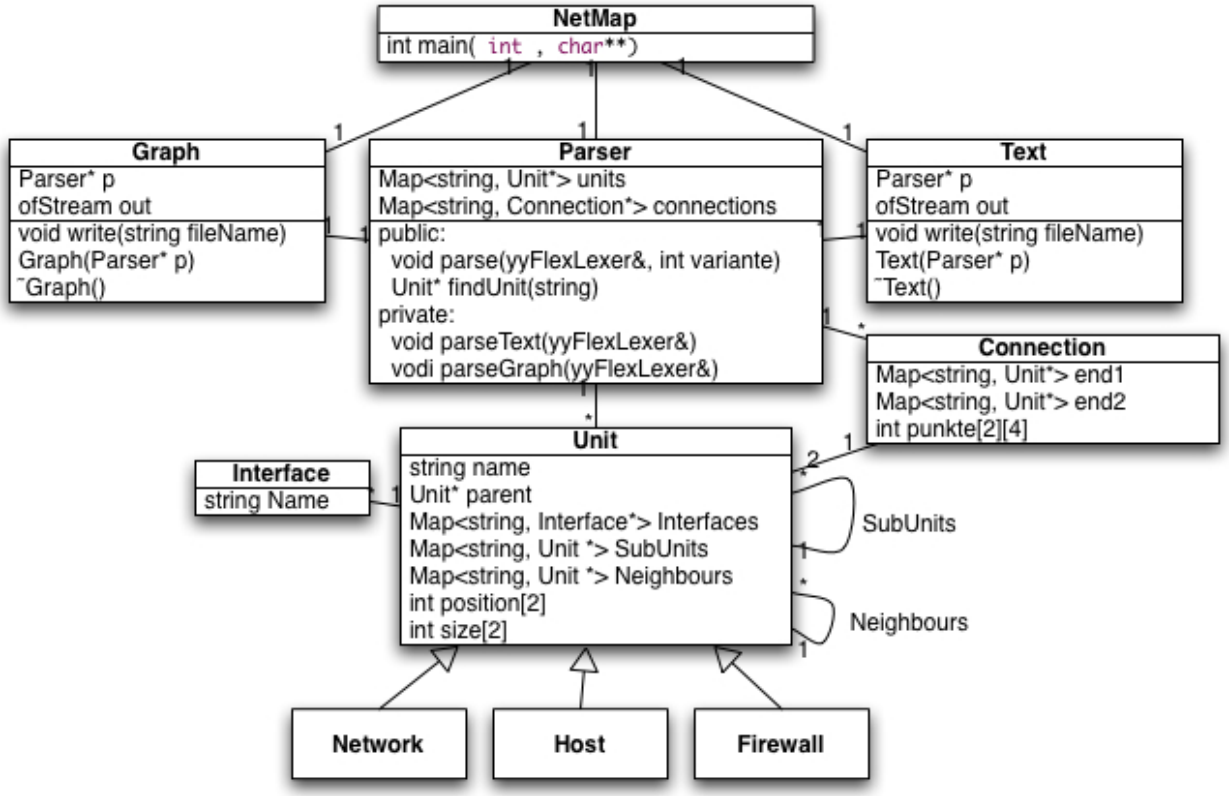

Abbildung 2: Implementation von NetMap

4.5.1. unit.{h,cpp}

Beschreibt die einzelnen Units eines Netzwerkes.

Ein Unit enthält Eigenschaften, die seinen Namen, die Services, welche dieses Unit anbietet sowie die 'Annotations' des Units beschreiben. Des weiteren kennt ein Unit seine Position und Grösse. Ausserdem weiss es, wer seine Nachbarn und SubUnits sind. Es kennt zudem, falls es selbst ein SubUnit ist, seinen Vater. Auf alle diese Eigenschaften kann mit get- und set-Methoden zugegriffen werden.

Ausserdem weiss ein Unit, welche Interfaces es hat. Auch diese Information ist durch geeignete Methoden zugreifbar.

Ein Unit kann positioniert werden. Wenn das geschieht, positioniert das Unit alle seine Nachbarn, die noch nicht positioniert wurden.

Ein Unit kann sich selber in verschiedenen Formaten ausgeben:

- Als einfachen Text zu Debugging-Zwecken.
- Im mfr-Format ⇒ Textausgabe.
- Im ssd-Format ⇒ Graphikausgabe.

Network : Unit Erbt von Unit. Im Moment noch weitestgehend ungenutzt.

Firewall : Unit Erbt von Unit. Im Moment noch weitestgehend ungenutzt.

Host : Unit Erbt von Unit. Im Moment noch weitestgehend ungenutzt.

4.5.2. connection.{h,cpp}

Beschreibt die Verbindungen (Connections) zwischen den Units des Netzwerkes.

Eine Connection weiss, welche Units an ihren Enden liegen. Sie kennt zudem ihre eigene Annotation. Auf alle diese Attribute kann mit get- und set-Methoden zugegriffen werden. Ausserdem weiss eine Connection ein bisschen was zur eigenen Geometrie. Diese Geometrieinformationen kann eine Connection sich selber aus den Informationen, welche zu den beiden Endpunkten besteht errechnen.

Eine Connection kann sich selber in den gleichen Formaten ausgeben, wie dies auch ein Unit kann.

4.5.3. interface.{h,cpp}

Beschreibt die Netzwerkinterfaces eines Units.

Ein Interface kennt ein paar wenige Attribute, die es beschreiben:

- Name
- ipRange (welcher IP-Bereich kann von diesem Interface sinnvollerweise angenommen werden).
- Property (eine einfache Beschreibung des Interfaces).
- Netz dahinter (eine textuelle Beschreibung des Netzes, zu dem dieses Interface eine Verbindung hat).

Auch hierfür sind die notwendigen set- und get-Methoden vorhanden.

Ausserdem kann sich ein Interface auf die bereits beschriebenen Arten, analog Unit und Connections, ausgeben.

4.5.4. parser.{h,cpp}

Erledigt das Parsen sowie das Erstellen der internen Repräsentation des Netzwerkes.

Der Parser enthält als einen zentralen Punkt alle Units und Connections. Seinen Input bekommt er vom automatisch erstellten Scanner in Form von Tokens.

Die wichtigste Methode des Parsers ist `void parse(yyFlexLexer&, int variante)`. Sie macht allerdings bloss einen Aufruf von `parseText` beziehungsweise `parseGraph` je nachdem welche Variante angegeben wurde. Diese Methoden wiederum parsen dann die erhaltenen Tokens und erstellen daraus die entsprechende interne Repräsentation.

4.5.5. scanner.{h,cpp}

Beschreibung, wie die beiden Inputfiles (Output von tssd und das Textfile) in Tokens aufgeteilt werden sollen.

Diese Files werden von flex automatisch erstellt. Als input hierzu dient `mfr.l`.

4.5.6. NetMap.cpp

Besteht genau aus der main-Methode und ist die eigentliche Implementation des Control-Flows (Figure 3).

Damit NetMap richtig funktioniert benötigt es drei Parameter:

1. Einen Integer: die Variante, welche ausgeführt werden soll: Text ⇒ Graph oder umgekehrt.
2. Das InputFile.
3. Das OutputFile.

4.5.7. Weitere Files

Graph.{h,cpp} Kümmert sich um die Ausgabe eines eingelesenen Netzwerkes in ein Format, das `tssd` verstehen wird.

Text.{h,cpp} Kümmert sich um die Ausgabe eines eingelesenen Netzwerkes in ein Format, das dem beschriebenen Textformat entspricht.

Token.{h,cpp} Zentral an der Klasse Token ist der Enum, welcher auch hier beschrieben ist. Darin sind alle in NetMap verwendeten Tokens aufgeführt (und noch ein paar mehr, die evt. in einer späteren Version verwendet werden können).

misc.{h,cpp} Enthält Methoden um einen String in einen Integer zu konvertieren und umgekehrt.

4.6. Netzwerkbeschreibung

Wie im Design schon beschrieben müssen in einem Netzwerk die Knoten, deren Verbindungen sowie die Eigenschaften der beiden beschrieben werden. Dies geschieht entweder in `tssd` indem man sie zeichnet, Annotationen dazu macht und den einzelnen Objekten einen Namen gibt oder in textueller Form in dem, von mir neu erstellten, Fileformat `mfr`.

4.6.1. Control Flow

Der Ablauf von NetMap ist in Abbildung 3 dargestellt.

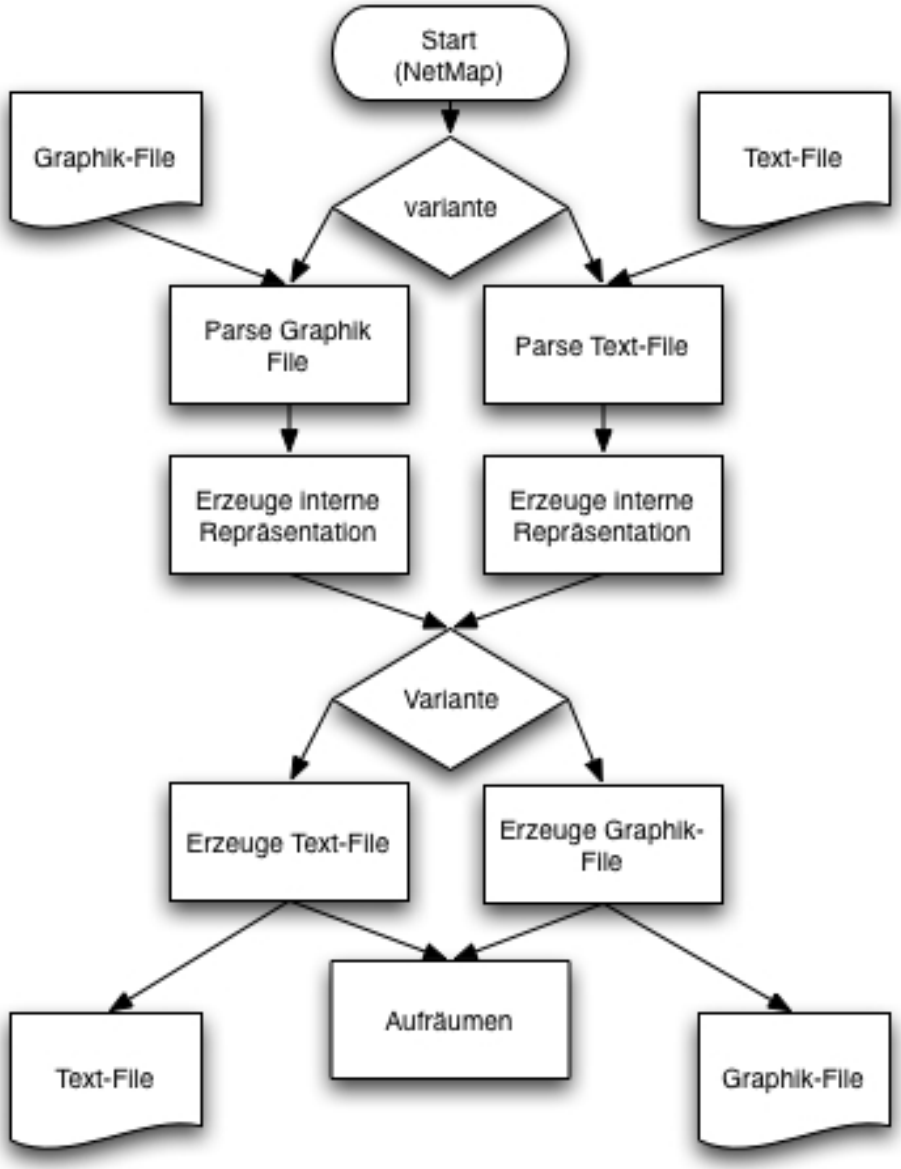

Abbildung 3: Control Flow von NetMap

5. Beispiel

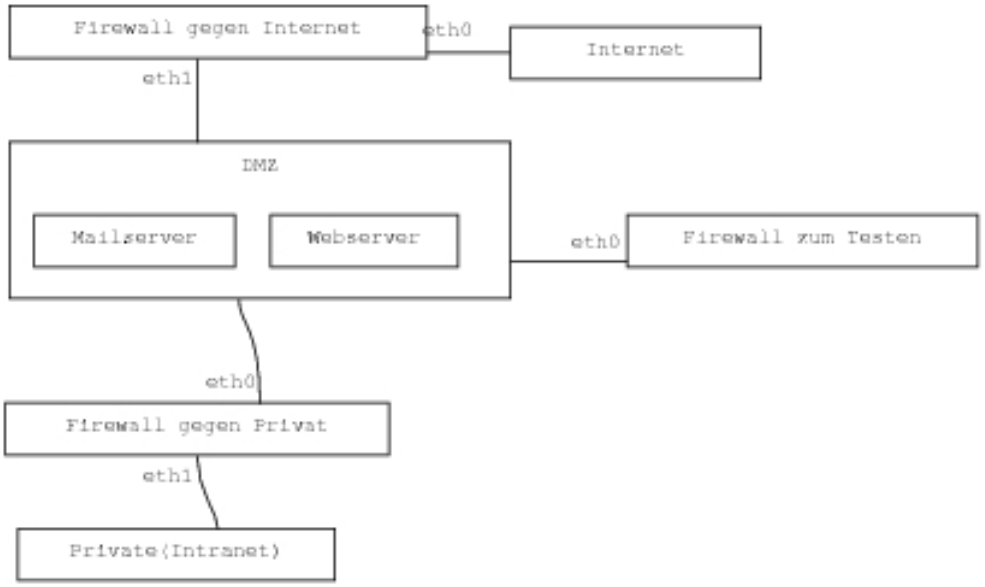

Abbildung 4: Möglicher Input eines Testnetzes

Aus dem mit tssd erstellten Input von Abbildung 4 wird mit NetMap das folgende Text-File erstellt:

```
// Erstellen von Einheiten
add( Network,    web,   "Internet" , );
add( Network,    dmz,   "DMZ" , );
add( Network,    priv,  "Private(Intranet)" , );
add( Firewall,   fw1,   "Firewall gegen Internet" , );
add( Firewall,   fw2,   "Firewall gegen Privat" , );
add( Firewall,   fw3,   "Firewall zum Testen" , );

// Erstellen von Sub-Eiheiten
subAdd( Host,  ms,     "Mailserver",  dmz,  "smtp, imap");
subAdd( Host,  ws,     "Webserver",   dmz,  "http");

// Erstellen von Interfaces
ifAdd( web, none, ,      ,      );
ifAdd( dmz, none1, ,     ,      );
ifAdd( dmz, none2, ,     ,      );
ifAdd( dmz, none3, ,     ,      );
ifAdd( priv, none, ,     ,      );
ifAdd( fw1, eth0, 0.0.0.1,          -,          "Internet und so");
ifAdd( fw1, eth1, 129.132.178.193,  -,              "DMZ");
ifAdd( fw2, eth0, 129.132.178.194,  Packet_filter,  "DMZ");
ifAdd( fw2, eth1, 129.132.178.197,  Packet_filter,  "Privat");
ifAdd( fw3, eth0, 129.132.178.194,  Packet_filter,  "keines");

// Erstellen von Verbindungen zwischen den Einheiten
```

```
connection( webfw1,    web,     fw1,     none,    eth0   );
connection( fw1dmz,    fw1,     dmz,     eth1,    none1  );
connection( dmzfw2,    dmz,     fw2,     none2,   eth0   );
connection( fw2priv,   fw2,     priv,    eth1,    none   );
connection( fw3dmz,    fw3,     dmz,     eth0,    none3  );

// Erstellen von Beschreibungen zu Einheiten oder Verbindungen
annotationAdd( dmz,     "Demilitarisierte Zone. Zweizeiliger\rText");
annotationAdd( fw1,      "Policy der Firewall gegen das Internet");
annotationAdd( fw1,      "ein zweiter Kommentar zum gleichen");
annotationAdd( webfw1,   "sogar Verbindungen kann man beschwafeln");
```

Wenn eben dieses Text-File wiederum mit NetMap in ein Graphik-File umgewandelt wird, so entsteht in `tssd` Abbildung 5

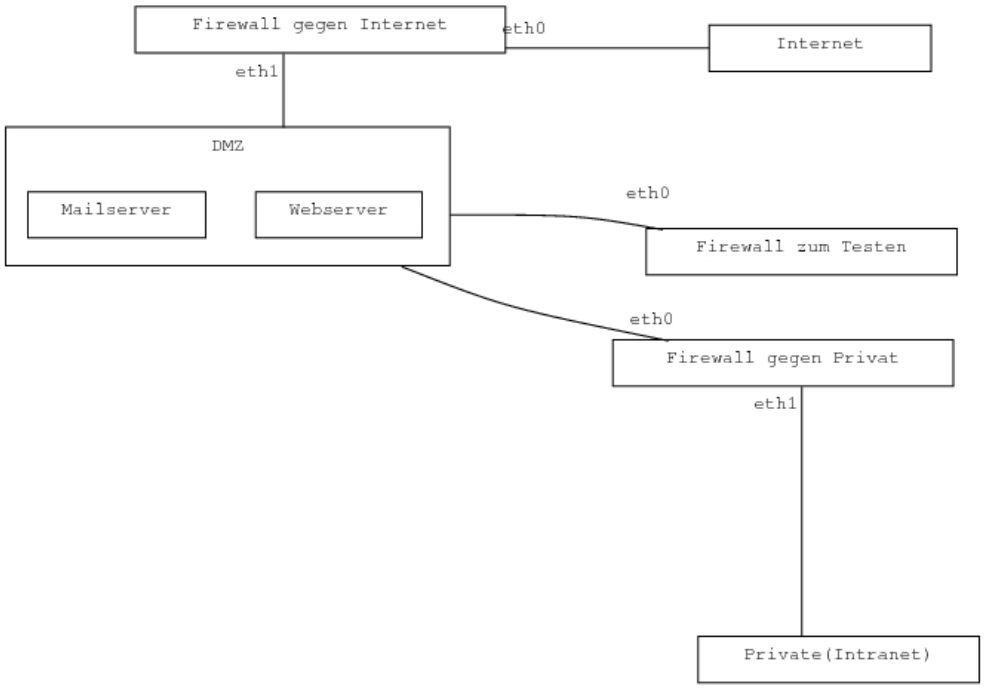

Abbildung 5: Möglicher Output eines Testnetzes

In der Handhabung ist NetMap sehr einfach. Es handelt sich um ein Kommandozeilentool. Im Moment ist NetMap noch ein bisschen wählerisch, was seinen Input anbelangt. Das sollte aber eigentlich keine Probleme machen, da die eine Seite von `tssd` erstellt wird und die andere Seite wohl entweder maschinell erstellt wird oder aber eher als Input für weitere Programme dient. Es ist also nicht geplant, dass ein normaler Anwender Dateien des Formates `.mfr` von Hand erstellt und NetMap damit füttert.

6. Fazit und Ausblick

6.1. Probleme

Schwierigkeiten tauchten relativ wenige auf. Hier ein Auszug der Wichtigsten:

- Schwierigkeiten mit C++ und dessen Eigenheiten, wie zum Beispiel die sichere Umwandlung eines Strings in einen Integer.
- Platzierung der Units (siehe 4.4).
- Versionsproblematik bei flex: Der Lexergenerator flex kommt auf verschiedenen Systemen in Versionen vor, die starke Unterschiede aufweisen. So hat die Version 2.5.4a, die auf den meisten Systemen installiert ist, einen ganz wichtigen Parameter zu wenig: man kann kein Headerfile angeben, das geschrieben werden soll. Das kommt wohl daher, dass im Normalfall die main-Methode gleich in dem Lexer selbst geschrieben wird, beziehungsweise, dass flex eher selten ohne ein Parsergenerator, der dann die nötigen Schritte kann, verwendet wird.
- Motivation vs. Zeitplan: Schon nach kurzer Zeit war ich dem Zeitplan massiv voraus, was dazu geführt hat, dass ich die Dokumentation zu spät in Angriff genommen habe. Als Folge davon bekam ich eine Überschneidung mit späteren Arbeiten, die ich erledigen musste, was zu einer massiven Verspätung der Abgabe führte.

6.2. Fazit

6.2.1. Was ich aus dieser Semesterarbeit gelernt habe

Ich glaube mittlerweile ein ganzes Stück besser C++ programmieren zu können. Was nebenbei eines meiner persönlichen Ziele für diese Semesterarbeit ist.

Trotz der Vorlesung Compilerdesign, welche ich besucht hatte, fühlte ich mich noch nicht wirklich sattelfest, was das scannen und parsen von Dateien anbelangt. Diesen 'Mangel' konnte ich dank dieser Semesterarbeit auch noch ausmerzen.

Mir war bis zum Moment, als ich mir darum in dieser Arbeit Gedanken machen musste, nicht klar, wie komplex es ist Knoten eines Netzwerkes so zu platzieren, damit sie nicht aufeinander liegen und damit sich die Verbindungen nicht zu stark überschneiden.

Dass Firewalls und ihre Konfiguration nicht einfach sind, war mir schon vor der Arbeit klar. Dass es aber Sinn machen könnte, die Policies einer Konfiguration von Firewalls `automatisch` zu testen, auf diese Idee wäre ich wohl nicht so einfach gekommen.

6.2.2. Macht das Resultat dieser Semesterarbeit Sinn?

Für mich persönlich, ganz aus dem Zusammenhang des Firewall testings rausgerissen, liegt der Sinn des Resultates unter anderem auch in den Dingen, welche ich gelernt habe. Insofern macht diese Arbeit sogar viel Sinn.

Im Zusammenhang des Firewall testings möchte ich bezweifeln, ob die gesamte Arbeit sinnvoll war. Im Moment ist es zwar möglich eine graphische Repräsentation in eine texuelle umzuwandeln, was das Ziel der Arbeit war. Wenn aber der nächste Schritt des Gesamtablaufes inklusive dessen zu erwartender Input bekannt gewesen wäre, so hätte ich das Outputformat von NetMap dem Input dieses nächsten Schrittes anpassen können. Oder aber zumindest den Output soweit vorbereiten können, damit er einfach auf das neue Format anpassbar gewesen

wäre. Ich habe aber troztdem versucht, den Output derart einfach zu gestalten, dass er mit minimalem Aufwand an ein neues, noch zu definierendes Format anpassbar ist.

6.3. Ausblick

Eine Arbeit, die in einer beschränkten Zeit ausgeführt wird, weist wohl immer noch Verbesserungspotential, beziehungsweise Punkte, die man noch erweitern kann, auf. Bei NetMap ist das nicht anders.

Hier die Punkte, die noch verbessert / geändert werden könnten:

Geometrie-History Wird im Moment der Zyklus Graphik ⇒ Text ⇒ Graphik durchlaufen, dann gehen im Zwischenschritt mit dem Text jegliche Geometrie-Informationen verloren. Das ist natürlich unangenehm, denn eventuell hat man ein komplizierteres Netzwerk, das speziell dargestellt werden muss, um verständlich zu sein. Oder aber man will mit einer speziellen Darstellung gewisse Sachverhalte betonen.

In beiden Fällen wäre es eine vergebene Mühe, die Netzwerke speziell schön zu zeichnen, da, wie gesagt, die geometrischen Informationen verloren gehen.

Um das zu verbessern, müsste man die Geometrieinformationen, die in dem Graphik-File drin sind auch noch mitparsen und in die IR schreiben. Dabei ist die IR bereits auf einen solchen Schritt vorbereitet: sie kann die Geometrieinformationen eines Units (und auch einer Connection) speichern.

Fileerkennung Zur Zeit ist es nötig NetMap einen Parameter mitzugeben um zu wissen, ob aus Text Graphik werden soll oder umgekehrt. Dies obschon das Format der beiden Inputfiles einfach vom Lexer erkannt wird. Man könnte also auf Grund des ersten Tokens entscheiden, ob es sich um ein Graphik- oder Text-File handelt.

Platzierung der Units Platzieren von Knoten und legen deren Verbindungen in Netzwerken ist kein einfaches Problem. Es wurde unter anderem in [1] gelöst. Aber genau dort ist auch beschrieben, dass ein Netzwerk, wenn es zweimal gezeichnet wird nicht zwingend gleich aussehen muss. Offensichtlich wir ein nicht deterministischer Alogrithmus verwendet. Also eine nicht wirklich befriedigende Lösung.

Bei dem einfachen Algorithmus, den ich verwendet habe, kann es vorkommen, dass einzelne Units auf anderen zu liegen kommen. Ausserdem kann es durchaus vorkommen, dass sich viele Connections überkreuzen. Diese beiden Tatsachen sind nicht wirklich schlimm, aber doch unangenehm. Es wäre wünschenswert in einer späteren Version einen besseren Algorithmus zu implementieren, der diese Nachteile beseitigt. Eventuell kann dazu direkt [1] verwendet werden.

Subnetze Eventuell könnte es in Zukunft auch Sinn machen, Netze, welche aus Subunits bestehen komplexer zu gestalten. Diese zu verbinden und auch noch detaillierter zu beschreiben.

A. Zeitplan

#	Title	Net work	Start date
	NetMap	**25 days**	**2005-03-29 9:00 AM**
1	**Vorbereitung**	**2 days**	**2005-03-30 8:00 AM**
2	Aufbau Umgebung	1 day	2005-03-30 8:00 AM
3	Planung	1 day	2005-03-31 9:00 AM
4	Suche und Evaluation	3 days	2005-04-01 9:00 AM
5	**Implementation**	**13 days**	**2005-04-07 9:00 AM**
6	Definition schoener Tex	1 day	2005-04-07 9:00 AM
7	**UML**	**4 days**	**2005-04-08 9:00 AM**
8	Text -> Graphic	2 days	2005-04-08 9:00 AM
9	Graphic -> Text	2 days	2005-04-13 9:00 AM
10	**Code**	**8 days**	**2005-04-15 9:00 AM**
11	Text -> Graphic	4 days	2005-04-15 9:00 AM
12	Graphic -> Text	4 days	2005-04-22 9:00 AM
13	Implementation abgeschlo:		2005-05-02 1:00 PM
14	**Nachbearbeitung**	**7 days**	**2005-04-29 9:00 AM**
15	Testen	3 days	2005-04-29 9:00 AM
16	Praesentation	1 day	2005-05-05 9:00 AM
17	Abschliessen Doku	3 days	2005-05-06 9:00 AM

B. Read Me von NetMap

```
Anleitung zur Verwendung von NetMap
-----------------------------------

NetMap <Nr> <inputFile> <outputFile>

Nr: welche Variante soll gemacht werden:
  1: Text -> Graphik: Erzeugt aus einem Texfile des Formates mfr
     ein GraphikFile des Formates ssd.
  2: Graphik -> Text: Erzeugt aus einem GraphikFile des Formates
     ssd ein Texfile des Formates mfr.

inputFile:
  1: Ein File des Formates mfr.
  2: Ein File des Formates ssd (erzeugt von tssd).

outputFile:
  1: Ein File des Formates ssd (erzeugt von tssd).
  2: Ein File des Formates mfr.


Anleitung fuer tssd fokussiert auf den Input fuer NetMap
========================================================

Erzeugen von Units ('Class') und Verbindungen:
----------------------------------------------

- Da NetMap nur mit 'Class' umgehen kann, muss auch diese Variante
  gewaehlt werden (default).
- Damit die Verbindungen ein bisschen schoener werden empfiehlt es
  sich bei den 'edges' den CheckButton 'createCurve' anzuwaehlen.
- Links-klick erzeugt eine neue 'Class'. Ein direkt darauf
  folgender Links-klick (in die erstellte 'Class') ermoeglicht das
  Editieren des Namen der erzeugten 'Class'
- Mittel-klick (halten) auf eine 'Class' erzeugt den Beginn einer
  Verbindung. Das erste Mal loesen der Taste erzeugt den zweiten,
  von vier moeglichen / noetigen Punkten. Ein weiterer Mittel-
  klick erzeugt den dritten Punkt. Der letzte Mittel-klick (
  innerhalb der Ziel-'Class') erzeugt den Endpunkt.
- Links-klick an der richtigen Stelle, unmittelbar nach der
  Erzeugung einer Verbindung ermoeglicht das Editieren der
  Informationen zu einer Verbindung. Richtige Stellen sind hierbei
  : links / rechts bei Beginn / Ende der Verbindung. Wird an einem
  dieser Orte geklickt erscheint in der Statusanzeige von tssd
  der Name des Feldes, welches editiert werden soll. Es gibt vier
  (als fuenfter: der Name der Verbindung: wird hier nicht
  verwendet) moegliche Felder, die man editieren kann. Dabei macht
```

```
    allerdings bloss das Feld 'Role Name' Sinn (alles andere wird
   von NetMap nicht uebernommen).

Erzeugen von Interfaces in tssd
-------------------------------

Da Verbindungen auch existieren koennen (sogar auch Sinn machen)
   ohne die genauere Beschreibung von Interfaces, ist das Erzeugen
   von Interfaces unabhaengig von den erzeugten Verbindungen.
   Ausserdem sind die Interfaces eine Eigenschaft eines Units und
   nicht die einer Verbindung.
Ein Interface wird streng genommen also nicht erzeugt, sondern
   bloss beschrieben (trotzdem werde ich beide Ausdruecke verwenden
    um den Prozess zu beschreiben). Diese Beschreibung geschieht in
    der 'Annotation' zum Unit:
1. Anwaehlen des Units, zu welchem das Interface hinzugefuegt
   werden soll.
2. Unter 'Properties' den Punkt 'Node/Edge Annotation' auswaehlen.
3. Da diese Annotation als String eingelesen und wiederum geparst
   werden muss, muss auch sie, genau so wie ein Inputfile des Types
    mfr, einer strengen Syntax gehorchen!
  IF: <Interface Name>, <IpRange>, <Eigenschaften>, [Netz dahinter
     ].
  Dabei ist zu beachten:
    - Die Felder, welche mit '<' '>' geklammert sind, sind Muss-
       Felder
    - Die Felder, welche mit '[' ']' geklammert sind, sind optional
    - soll einfach nichts da stehen, so ist ein '-' zu schreiben
    - '"' in den Feldern werden entfernt!
    - Am Schluss steht ein Punkt '.'
    - Am Schluss _muss_ mit einem NewLine abgeschlossen werden.
    - Jeglicher Text, der auf den Zeilen vor oder nach diesen
       Interfacebeschreibungen steht, wird als Annotation zu dem
       entsprechenden Unit interpretiert

Annotations
-----------

Sowohl Units wie auch Connections koennen annotiert werden. Dabei
   ist die zu annotierende Einheit auszuwaehlen und dann unter '
   Properties' der Punkt 'Node/Edge Annotation' auszuwaehlen. Nun
   kann allgemein normaler Text eingegeben werden. Dabei ist zu
   beachten, dass Text, der auf ein "#" folgt ignoriert wird.
   Ausserdem wird NetMap versuchen einen Text, welcher mit "IF:"
   auf den ersten drei Zeichen einer Zeile beginnt zu parsen, wie
   oben beschrieben. Wenn das also nicht explizit gewuenscht ist,
   sollte es unterlassen werden, da es sonst zu unvorherzusehendem
   Verhalten von NetMap fuehren kann.
```

Literatur

[1] John Ellson et. al. graphviz. www.graphviz.org.

[2] Vern Paxson et al. flex. http://www.gnu.org/software/flex/.

[3] icewalkers.com. tools. www.icewalkers.com/Linux/Software/Multimedia/Graphics/Editors/1630/.