Leopold-Franzens-Universität Innsbruck

Institut für Informatik

Forschungsgruppe Verteilte und Parallele Systeme

Ein graphischer Editor für ns-2 NSBM

Bakkalaureatsarbeit

eingereicht bei Dr.-Ing. Michael Welzl

Robert Binna

Wolfgang Gassler

Thomas Gatterer

Innsbruck, 07.06.05

Danksagung

Vielen Dank an Herrn Dr.-Ing. Michael Welzl, der diese interessante Bakkalaureatsarbeit angeboten hat und uns durch sein Wissen und Einsatz begleitet hat. Erwähnt sollen hier auch seine motivierenden Mails werden, mit deren Inhalt wir den richtigen Weg zu einem guten Ergebnis gefunden haben.

Danken möchten wir auch Sven Hessler, Muhammed Ali und Murtaza Yousaf, die NSBM als erste getestet haben und so manche Zeit investierten, um Fehler im NSBM zu finden.

Für das Korrekturlesen möchten wir Irene Pescolderung und Sabine Bertagnolli danken.

Inhaltsverzeichnis

1. Einleitung

1.1. Warum Netzwerksimulation

Paketorientierte Netzwerke haben im Laufe der letzten Jahrzehnte ungeahnte Dimensionen angenommen. Die Technologien, die diese Netzwerke aufrecht erhalten, werden ständig verbessert und erneuert. Im gleichen Maße ist auch die Komplexität der Netzwerkkomponenten gestiegen.
Obwohl die Netzwerke ständig wachsen, hat die steigende Menge an Daten, die durch die Verbindungen geschickt werden, den Datenfluss nicht ins Stocken gebracht. Um solchen Anforderungen gerecht zu werden, beschäftigt sich ein großer Teil der Netzwerkforschung mit dem Verändern von Parametern und Einstellungen von Netzwerken. Verschiedene Netzwerkszenarien müssen in kurzer Zeit untersucht und ihr Nutzen bewertet werden.
Datenpakete in einem Netzwerk werden auf verschiedenste Weise versendet, zwischengespeichert und weiter geschickt. Auf dem Weg den Pakete zurücklegen, können durch verschiedene Protokolle und Zwischenstationen viele verschiedene Ereignisse auftreten. Pakete werden verworfen und müssen neu geschickt werden. Pakete warten in Warteschlangen, bis sie weitergeleitet werden. Leitungen sind überfüllt und können keine zusätzlichen Lasten mehr aufnehmen. Alle diese Erscheinungen will man optimieren.
Netzwerkkomponenten verhalten sich dabei sehr unterschiedlich: Sender beginnen zur unterschiedlichen Zeiten zu senden oder Leitungen fallen zu nicht vorhersehbaren Momenten aus.
Diese Eigenschaften erschweren das Bewerten von Netzwerken. Ohne Simulation sind komplizierte, komplexe Zusammenhänge im Netzwerk nicht mehr erfassbar.
Simulieren bedeutet, dass gegebene Rahmenbedingungen am Rechner nachgebildet werden. Veränderungen an einem solchen Modell haben keinen Einfluss auf das bestehende Netzwerk. Somit kann man Störungen oder Ähnliches nachbilden, ohne den reellen Datenverkehr zu stören. Simulationen können mit ähnlichen Bedingungen wiederholt, verglichen und bewertet werden, ohne dass unvorhergesehene Einflüsse die Messungen stören.
Zukünftige Technologien müssen auf ihren Nutzen und ihre Funktionalität untersucht werden. Um neue und abgeänderte Protokolle zu testen und sie auf Fehler zu untersuchen, können Simulationen einen guten Nutzen leisten. Simuliert werden können beispielsweise Szenarien, in denen Netzwerkkomponenten ausfallen und Daten über alternative Wege umgeleitet werden müssen. Diese "was passiert wenn"- Szenarien

verdeutlichen, was in Ernstfällen passieren könnte und ermöglichen Überlegungen, wie man Ausfällen zuvorkommen könnte.
Simulationen liefern eine Entscheidungsgrundlage für das IT-Management. Beim Bewerten von Netzwerken ist man mit Simulationen nicht mehr nur auf Erfahrungen und Abschätzen angewiesen. Bei größeren Investitionen bietet die Simulation einen Schutz vor Fehlplanungen. Kosten bei Investitionen werden kalkulierbar, da der Nutzen besser berücksichtigt werden kann.
Risikoabschätzung bei Veränderungen im Netzwerk können besser vorbereitet werden.

1.2.Programme zur Netzwerksimulation

Durch die schnelle Entwicklung der Netzwerktechnologie wurden viele Simulationsprogramme geschrieben und erweitert. Die meisten davon sind auf einen bestimmten Netzwerktyp spezialisiert oder an ein vorhandenes Netzwerk angepasst.
Die Programme unterscheiden sich auch stark in ihrer Benutzung und Funktionalität. Einen Überblick über die am häufigsten verwendeten Tools kann man unter der Liste von Simulations Werkzeugen von Andrea Emilio Rizzoli [1] finden.
Hier wird nur auf die zwei wichtigsten eingegangen: *Opnet* [2] und *ns-2*.
Opnet bietet eine Reihe kommerzieller Netzwerk Simulations- und Analyseprogramme, die für die Entwicklung und das Design von Telefonnetzwerken, Netzwerkkomponenten, Protokollen und Netzwerkprogrammen dienen. Auch für die Untersuchung von bestehenden Netzwerken und deren Optimierung hat Opnet geeignete Tools.
Ns-2 (Network Simulator) ist hingegen ein frei erhältlicher Netzwerk Simulator, der gut erweiterbar und weit verbreitet ist. Er ist eine Weiterentwicklung des *REAL network simulators* [3] und wird vorwiegend in der Forschung eingesetzt.
Ns-2 ist eine Sammlung mehrerer Programme, die die Aufgaben eins Simulators abdecken. Das Programm ns ist der Netzwerksimulator, der die eigentliche Simulation des Netzwerkes übernimmt und später noch genauer beschrieben wird. Ns erzeugt Trace Files, in denen alle wichtigen Daten enthalten sind. Zum Auswerten der Trace Files kann man das Visualisierungswerkzeug nam (Network Animator) verwenden oder selber Scripte schreiben, die die Daten der Trace Files für andere Visualisierungswerkzeuge verfügbar machten.

1.3.Installation von ns-2

Um grundlegende Simulationen durchführen zu können, reicht im Normalfall das „all in one“ Paket, das man unter der „ns building“ Seite [4] finden kann. Dieses „all in one“ Paket beinhaltet neben dem ns und dem nam einige weitere Programme, die entweder vom ns oder nam benötigt werden oder zur weiteren Visualisierung der Daten dienen.
Bei der Frage, wie man zu neueren Versionen der in diesem Paket enthaltenen Programme kommt und welche Abhängigkeiten bestehen, wird hier auf die ns Projekt Homepage [5] verweisen. Dort sollten die neuesten Versionen der Programme oder deren Adressen zu finden sein.

1.4.ns

Der Aufruf des ns

Ns ist ein Kommandozeilen Programm, das aber auch als Interpreter genutzt werden kann. Ns benötigt zum simulieren ein OTcl Script, in dem die zu simulierende Netzwerktopologie spezifiziert ist. Dieses Skript wird beim Aufruf übergeben. Wird ns ohne Skript gestartet, befindet man sich in der Shell vom ns und kann dort einzeln Befehle eingeben.
Hat ns das OTcl Script oder die Befehle in der Shell abgearbeitet, erzeugt der Simulator eine Datei, in der alle wichtigen Simulationsdaten enthalten sind (der Befehl zur Erzeugung dieser Datei wird im OTcl Script oder in der ns Shell abgesetzt).

Die Script Datei

Im OTcl Script müssen alle Angaben des zu simulierenden Netzwerkes enthalten sein. Zu diesem gehören auch die Protokolle, die Ereignisse und die Information wohin Pakete geschickt werden sollen. Ereignisse sind z.B. der Beginn oder das Ende einer Übertragung.
Auch der Name der Ausgabe Datei des ns ist im Script enthalten.
Wie man für eine Netzwerktopologie ein Script schreibt, ist in diesem Dokument nicht angeführt, da man für die grundlegende Benutzung des NSBM dieses Wissen nicht benötigt. Für einen weiterführenden Einsatz der Netzwerksimulationstools ist das Verstehen, Abändern und Erweitern dieser Scripts jedoch unerlässlich. Deshalb wird hier auf die bereits bestehenden Dokumentationen verwiesen:
Das Marc Greis Tutorial [6] (Einstieg in Englisch).
Eine weiterführende deutsche Dokumentation [7] inklusive des übersetzten Greis Tutorials und das ns Benutzerhandbuches [8].

Die Ausgabedatei

Wie bereits erwähnt muss eine Ausgabedatei im Script spezifiziert werden. Alle simulationsrelevanten Informationen werden dort gespeichert. Diese erzeugten Dateien werden als Trace Files bezeichnet. Das Format der Datei wird im Skript spezifiziert; es stehen dabei mehrere Möglichkeiten zur Auswahl. Die am häufigsten verwendeten Formate werden mit dem Befehl „trace-all“ und „namtrace-all“ erzeugt. Der wesentliche Unterschied besteht darin, dass bei der namtrace-all Variante zusätzliche Informationen über die Netzwerkstruktur und andere Angaben, die nam (Network Animator) zum animieren benötigt mitprotokolliert werden.

Die vom trace-all erzeugte Datei ist auch wesentlich kürzer als die nametrace-all Datei. Die trace-all Datei kann man im Wesentlichen dazu benutzen, Berechnungen oder Grafiken zu erstellen. Dabei wird meist ein weiteres Script verwendet, das die Informationen so aufarbeitet, dass sie von einem Visualisierungswerkzeug verarbeitet werden können.

Die Funktionen des ns

Ns erfüllt zwei Aufgaben: die Simulation von paketorientierten Netzwerken, bei denen der Paket Header schnell und effektiv verändert werden muss. Zusätzlich müssen Bytes abgeändert werden und schnelle Netzwerkalgorithmen zur Verfügung gestellt werden. Diese Algorithmen müssen eine große Anzahl von Datensätzen in kurzer Zeit bearbeiten. Um dieser Aufgabe gerecht zu werden wird bei ns die Programmiersprache C++ verwendet.

Die zweite Aufgabe des ns ist es, Netzwerke zu simulieren, bei denen man Schlüsselparameter flexibel und schnell abändern kann, ohne dass ns neu übersetzt werden muss. Deswegen überlässt man das Einlesen der Netzwerktopologie (und deren dazugehörenden Parameter) einem (OTcl) Interpreter, der die gesamten Informationen zum zu simulierenden Netzwerk aus dem OTcl File liest.

In der Regel wird eine OTcl-Skript Datei erstellt, die vom ns simulator abgearbeitet wird. Diese Datei kann einfach verändert werden und dann von ns neu und schnell Interpretiert werden.

Dokumentation zu ns

Die umfangreichste Dokumentations-Liste zu ns findet man unter der Projekthomepage des Simulators [5], wobei das Marc Greis Tutorial den ersten Schritt darstellt. Übersetzungen für Benutzer in Deutsch findet man unter der ns Tool Seite [9]. In diesem Dokument ist auch die Übersetzung des oben genannten Greis Tutorials im Anhang enthalten.

1.5.nam

Nam wurde entwickelt, um die oben genannten Trace Dateien visualisieren und animieren zu können. Auch stellt nam die einfachste Variante dar, den Output des ns grob zu analysieren.
Der erste Schritt, um mit nam ein Netzwerk zu animieren, ist ein Trace File zu erstellen. Dies wird im Normalfall, wie oben beschrieben, mit ns erzeugt. Der Name des Trace File muss in der Kommandozeile beim Aufruf des nam angegeben werden. Nachdem nam gestartet wurde, liest es den ersten Teil des Trace Files ein (dort ist die Netzwerkstruktur enthalten) und zeichnet anschließend die Netzwerktopologie in ein Fenster. Danach wartet nam auf den Start der Animation. Die wichtigsten Funktionen die nam bereitstellt, um die Animation zu starten, zu stoppen, zurück und vor laufen zu lassen, sind einem Musik Player sehr ähnlich und werden hier nicht weiter erläutert. Näheres findet man dazu im Benutzerhandbuch des ns [8] im Abschnitt 9, Kapitel 42.3. Die gesamte Dokumentation des nam ist im Benutzerhandbuch des ns enthalten (Abschnitt 9 in dem Benutzerhandbuch des ns).

1.6.NSBM

a) Die Funktionen von NSBM:

- In erster Linie wurde NSBM erstellt, um grafisch Netzwerktopologien erstellen zu können, die in einem weiteren Schritt von ns simuliert werden können.
  Das von ns als Eingabe akzeptierte OTcl File wird von NSBM so generiert, dass es ohne eine weitere Bearbeitung von ns simuliert werden kann.
- Der Zeitpunkt bei dem ein Sender seine Pakete zu verschicken beginnt und anschließend wieder stoppt (Events), kann grafisch bestimmt werden. Man hat dabei einen Zeitbalken zur Verfügung, in dem alle Eventquellen aufgelistet sind und nach Belieben neue Events hinzugefügt oder gelöscht werden können. Dies bietet einen guten Überblick über alle auftretenden Events.
- Bei einigen Netzwerktopologien kann es nützlich sein, an einen Knoten mehrere andere Knoten zu hängen, die die gleichen Parameter haben (etwa bei den sogenannten „Dumbbell“ Topologien). Bei NSBM hat man die Möglichkeit einen Knoten stellvertretend für mehrere Knoten zu zeichnen (Multiple Knoten). Alle Knoten haben dann die gleichen Parameter. Die Anzahl der Knoten kann man als Attribut des Knotens spezifizieren.
- Welche Trace Files (nam Trace File oder reines Trace File, oder beide) ns aus dem OTcl Script erzeugt, kann angegeben werden. Auch die Namen

lassen sich im NSBM angeben.

- Um im generierten Code die erstellten Objekte leichter wieder zu finden, hat man die Möglichkeit, ihnen einen eigenen und eindeutigen Namen zu geben.
- Der generierte Code ist übersichtlich strukturiert und kommentiert.
- NSBM bietet die Möglichkeit globale Variablen zu erstellen. Diese können benutzt werden, um gleichzeitig Parameter von mehreren Netzwerkelementen zu verändern.
- Um verschiedene Netzwerkszenarien, bei denen sich Attribute ändern, schnell simulieren zu können, besteht die Möglichkeit Attribute beim Aufruf des ns als Parameter anzugeben.
- NSBM ist leicht zu erweitern: die Möglichkeiten die NSBM bereitstellt, hängen grundsätzlich nicht nur vom Programmcode des NSBM ab, sondern auch von XML files, die NSBM bei jedem Start einliest. In diesen XML files wird definiert, welche Netzwerk Elemente ns simulieren kann, welche Parameter sie besitzen und wie der Ausgabecode in OTcl aussehen soll. Die XML files sind leicht verständlich und können ohne großen Programmieraufwand abgeändert und erweitert werden.

b) Wie arbeitet man mit NSBM?

Ohne NSBM wird der OTcl Code in einem beliebigen Editor erstellt. Dieser Code wird ns übergeben und dann mit nam oder einem anderem Visualisierungswerkzeug analysiert. Das Erstellen von OTcl Code in einem Editor bringt einige Nachteile mit sich, die weiter unten angeführt sind. Als Alternative zur manuellen Erstellung des Codes wurde NSBM entwickelt; damit kann man die grundlegenden Aufgabenstellungen der Codeerstellung, grafisch durchführen. Dabei sind im Normalfall folgende Arbeitsschritte nötig:

- Grafisches Platzieren der Knoten und Anpassen ihrer Eigenschaften.
- Die erstellten Knoten mit Links verbinden.
- Die Queues der Links anpassen.
- Agenten (so werden die Protokolle genannt, die ein Knoten benützt - z.B. TCP, UDP, ...) hinzufügen und deren Eigenschaften anpassen.
- Festlegen, wohin die Agenten Daten verschicken (verbinden der Agenten).
- Applikation (so werden Datenproduzenten wie z.B. FTP und Telnet bezeichnet) erstellen. Anschließend deren Eigenschaften anpassen.
- Events erstellen (dies sind zeitlich abgegrenzte Ereignisse, in denen Applikationen senden) und deren Eigenschaften anpassen.

- Den Code generieren lassen.

Will man genauer spezifizieren, wie lange die Simulation dauern soll, welche Trace Files NSBM generieren soll, ob es gemeinsame Variablen geben soll oder ob Variablen beim Aufrufen des ns übergeben werden sollen, kann man dies zu jeder Zeit vor dem Generieren des Codes tun.

c) Gründe für NSBM?

Folgend sind einige Überlegungen zu den Vorteilen der „Grafischen Erstellung“ vom OTcl Code durch den NSBM angeführt:

- Beim Erstellen großer oder komplexer Netzwerktopologien entsteht innerhalb kurzer Zeit ein langer OTcl Code. Trotz einfacher Programmkonstrukte ist dieser oft unübersichtlich und kann daher schnell Fehler beinhalten. Tippfehler können dabei ebenfalls häufig auftreten. Eine grafische Erstellung bringt mehr Überblick, da man alle Netzwerkkomponenten gleichzeitig in intuitiver Form sehen kann. Auch Tippfehler treten nicht auf, da der Code automatisch generiert wird.
- Der Einstieg in die Netzwerksimulation ohne NSBM ist erschwert, da das Wissen zu einer nicht unbedingt üblichen Programmiersprache zum Darstellen einer Netzwerktopologie benötigt wird. Intuitiv wird nicht vermutet, dass man zum simulieren eines Netzwerks eine Programmiersprache benötigt. Diesen Einstieg erleichtert NSBM, da für grundlegende Netzwerke keine manuelle Code Erstellung oder Kenntnis über die Programmiersprache des Codes von Nöten ist.
- Nam bietet bereits die Möglichkeit grafisch Netzwerke zu zeichnen und daraus OTcl Code generieren zu lassen. Dieser nam editor wird jedoch in seiner Dokumentation als noch „under construction“ beschrieben. Neben der etwas umständlichen Benützung und der beschränkten Möglichkeiten des Editors ist auch die eingeschränkte Erweiterbarkeit zu erwähnen: der nam editor ist ebenfalls in OTcl geschrieben, wobei OTcl wenig verbreitet ist und man sich für Erweiterungen diese Programmiersprache aneignen müsste. NSBM ist in Java geschrieben, das neben den umfangreichen Möglichkeiten auch einen sehr hohen Bekanntheitsgrad aufweist. Auch Erweiterungen des ns bezüglich neuer Protokolle werden in NSBM durch XML Files definiert und sind leicht durchzuführen.

d) Einschränkungen des NSBM

- NSBM ist ein eigenständiges Programm. Somit werden Erweiterungen von ns nicht automatisch berücksichtigt. Ns Erweiterungen können solange in den XML Files des NSBM berücksichtigt werden, solange die vorhandenen OTcl Codestrukturen beibehalten werden.
- Neben den bereits verwendeten Programmiersprachen C++ für die

zeitkritischsten Funktionen des ns und OTcl für die Abstraktion der zu simulierenden Netzwerktopologie wird nun auch Java für den NSBM eingesetzt. Erweiterungen werden in XML Dateien spezifiziert. Für eine einfache Verwendung muss jedoch kein Wissen zu einer der genannten Sprachen verfügbar sein.

- Bei der Generierung des OTcl Codes werden nicht alle Möglichkeiten ausgeschöpft, die ns mit Hilfe von OTcl anbietet. Verschiedene Farben von Paketen oder weitere Möglichkeiten bei den Trace Files könnten als Beispiel angeführt werden. Teilweise würden aber diese Möglichkeiten die Bedienung des NSBM so erschweren, dass einer der zentralen Punkte, der einfache Einstieg in die Netzwerk Simulation, nicht mehr erfüllt würde. Auch ist anzunehmen, dass ab einem bestimmten Benutzungsgrad der OTcl Code so stark manuell im Editor weiter entwickelt wird, dass NSBM nicht mehr zum Einsatz kommt.
- Das Einlesen eines bereits bestehenden OTcl Codes ist bei NSBM nicht vorgesehen, da die verschiedenen Codevarianten für ein und dieselbe Netzwerktopologie zu umfangreich sind.

e) Wo gibt es Dokumentation zum NSBM?

Die NSBM Dokumentation ist als „documentation package“ gesammelt unter der ns Tool Seite [9] zu finden.

In diesem Package befinden sich:

- drei Beispieldateien (example1.nsm, example2.nsm, example3.nsm), die von NSBM geladen werden können.
- das FastTutorial.txt, eine Textdatei in englisch, in der die wesentlichsten Schritte zum Erstellen der drei oben genannten Beispiele beschrieben sind. Dieses Tutorial ist für Personen konzipiert, die bereits mit ns gearbeitet haben und dessen Konzepte verstehen.
- der aus den einzelnen Beispieldateien generierte Code (CodeExample1.tcl, CodeExample2.tcl, CodeExample3.tcl). Diese können mit den eigenen Ergebnissen aus dem „FastTutorial“ verglichen werden.
- das UserManual.pdf in Englisch. In diesem Dokument sind alle Funktionen vom NSBM erklärt: Installation, Erweiterung von NSBM mit Hilfe der XML Files und Erklärungen zum generierten Code.
- und schließlich dieses Dokument: die deutsche Dokumentation zu NSBM, mit der man NSBM ohne Vorkenntnisse nutzen kann.

2.Benutzerhandbuch

Im Benutzerhandbuch befindet sich die Beschreibung der Installation des NSBM sowie eine Anleitung zur grafischen Erstellung von Netzwerktopologien. Auch sind im Benutzerhandbuch Informationen enthalten, wie der generierte Code zu verstehen ist und wie man NSBM über die XML Files erweitern kann.
Da dies das am meisten benutzte Dokument zu NSBM sein wird, und die Benutzer sich nicht auf den deutschsprachigen Raum beschränken, wurde das Benutzerhandbuch in englischer Sprache verfasst. Es befindet sich im Anhang dieses Dokumentes und im documentation package.

3.Beispiele

In diesem Kapitel werden zwei Beispiele beschrieben, um die Arbeitsweise von NSBM im Zusammenhang mit den Netzwerksimulationstools zu beschreiben. Die genaue Lage der Bedienelemente erfährt man aus dem Benutzerhandbuch im Anhang. Der erzeugte Code und die Datei, die NSBM lädt, ist im documentation package enthalten. Dabei entspricht das example1.nsm und das codeExample1.tcl dem ersten Beispiel und das example3.nsm und codeExample3.tcl dem zweiten Beispiel.

3.1. Beispiel 1

Das erste Beispiel dient als Grundlage für den Einstieg in die Bedienung von NSBM und das Benutzen seiner Ausgabedateien. Das Ergebnis ist ähnlich dem ersten Beispiel aus dem Marc Greis Tutorial, wobei man von einer einfachen Netzwerktopologie zu einer Simulation in nam gelangen soll. Der Unterschied zum Greis Tutorial liegt darin, dass der OTcl Code von NSBM generiert wird, daher auch etwas von der handgeschriebenen Variante aus dem Greis Tutorial abweicht und die Topologie im NSBM grafisch eingegeben wird. Gleich ist hingegen die Ausgabe des nam.
Die Netzwerktopologie besteht aus zwei Knoten, an denen an der einen Seite ein UDP Agent und an der anderen eine Paket Senke (Null) stehen. Die Daten, die von einem Knoten zum anderen geschickt werden sollen, werden von einer CBR (Constant Bit Rate) Quelle erzeugt. Diese Quelle soll von Sekunde 0.5 bis Sekunde 4.5 Pakete schicken.

a) Schritte zum Zeichnen der Topologie

- Nach dem Start von NSBM wählt man aus dem *Tool* Fenster das *Node* Werkzeug aus (mit „N“ gekennzeichnet) und zeichnet mit ihm auf der Zeichenfläche zwei Knoten.
- Anschließend wählt man aus dem *Tool* Fenster das *Link* Werkzeug aus (mit „L“ gekennzeichnet) und verbindet die beiden Knoten.
- Um den ersten UDP Agenten zum ersten Knoten hinzuzufügen, klickt man mit der rechten Maustaste auf den Knoten und wählt dort *Add Agent->UDP* aus.
- Dasselbe beim zweiten Knoten, nur wählt man dort als Senke den Agenten „Null“.
- Um den Verkehrsgenerator CBR zu unserer Topologie hinzuzufügen, klickt man mit der rechten Maustaste auf den UDP Agenten und anschließend auf *Add Application->CBR*.
- Um festzulegen, wohin die Pakete geschickt werden sollen, muss man noch beide Agenten mit dem *Link* Tool verbinden.
- Um einstellen zu können, dass der CBR von 0.5 bis 4.5 sendet, muss man im Menu *Configuration->Session configuration* die Timeunit auf ½ stellen. Jeder Abschnitt in der Timeline entspricht dann einer halben Sekunde.
- Die gesamte Laufzeit der Simulation soll 5 Sekunden betragen, also muss im Menu *Configuration->Session configuration* die Runlength auf 10 gestellt werden (bei einer halben Sekunde pro Abschnitt ergibt das genau 5 Sekunden).
- Im "Timeline" Fenster mit der rechten Maustaste auf „TrafficCBR1“ und dort *New run event* auswählen, um anschließend mit dem Zeitbalken die richtigen Zeiten einzustellen.
- Anschließend kann im Menü *File->Generate* NS Code der OTcl code generiert werden. Dazu muss man Verzeichnis und Name der Ausgabe Datei angeben.

b) Der generierte Code

```
#############################################
# Tcl Script created by NSBM - Network Simulation by Mouse
# Copyright 2004
#
#
#
#############################################

set ns [new Simulator]

set nf [open my_output.nam w]
$ns namtrace-all $nf

proc finish {} {
  global ns nf tf
  $ns flush-trace

  close $nf
  #exec nam my_output.nam &
  exit 0
}

#############################################
#
# Procs to manage multiple nodes
#
#############################################

proc createMultipleNode {nodes number} {
  upvar $nodes n
  global ns
  for {set i 0} { $i < $number} {incr i} {
    set n($i) [$ns node]
  }
}

proc linkMultipleSingleNode {nodes node band delay queue} {
  upvar $nodes n_multiple
  upvar $node n_single
  global ns
  set x  [array size n_multiple]
  for {set i 0} { $i < $x} {incr i} {
    $ns duplex-link $n_multiple($i) $n_single $band $delay $queue
  }
}
proc linkMultipleMultipleNode {nodes1 nodes2 band delay queue} {
  upvar $nodes1 n_multiple1
  upvar $nodes2 n_multiple2
  global ns
  set x [array size n_multiple1]
  for {set i 0} { $i < $x} {incr i} {
    $ns duplex-link $n_multiple1($i) $n_multiple2($i) $band $delay $queue
  }
}

proc attachMultipleAgent {node agent} {
  upvar $agent agents
```

```
  upvar $node nodes
  global ns
  set x [array size agents]
  for {set i 0} { $i < $x} {incr i} {
    $ns attach-agent $nodes($i) $agents($i)
  }
}

proc attachApplicationAgent {app agent} {
  upvar $app apps
  upvar $agent agents
  global ns
  set x [array size apps]
  for {set i 0} { $i < $x} {incr i} {
    $apps($i) attach-agent $agents($i)
  }
}

proc connectMultipleAgent {agent1 agent2} {
  upvar $agent1 ag1
  upvar $agent2 ag2
  global ns
  set x [array size ag1]
  for {set i 0} { $i < $x} {incr i} {
    $ns connect $ag1($i) $ag2($i)
  }
}


##################################################


##################### Global Variables



##################### Nodes Code

set Node1 [$ns node]
set Node2 [$ns node]


##################### Links Code

$ns duplex-link $Node1 $Node2 1Mb 10ms DropTail
set Link1 [$ns link $Node1 $Node2]


##################### Queues Code

set DropTail1 [$Link1 queue]
 $DropTail1 set unblock_on_resume_ true; #dokuin
 $DropTail1 set limit_ 50; #dokuin
 $DropTail1 set blocked_ false; #dokuin


##################### Agents Code

set UDP1 [new Agent/UDP]; #null
```

```
 $UDP1 set packetSize_ 1000.0; #


set Null1 [new Agent/Null]; #null



$ns attach-agent $Node1 $UDP1
$ns attach-agent $Node2 $Null1


$ns connect $UDP1 $Null1

#################### Applications Code


set TrafficCBR1 [new Application/Traffic/CBR]; #null
 $TrafficCBR1 set random_ 0; #
 $TrafficCBR1 set interval_ 0.0050; #
 $TrafficCBR1 set rate_ 448Kb; #
 $TrafficCBR1 set packetSize_ 210.0; #
 $TrafficCBR1 set maxpkts_ 268435456; #




$TrafficCBR1 attach-agent $UDP1
#################### Time Events



$ns at 0.5 "$TrafficCBR1 start"

$ns at 4.5 "$TrafficCBR1 stop"

$ns at 5.0 "finish"

$ns run
```

c) Vom OTcl Code zur Simulation

- Um das Netzwerk zu simulieren, gibt man in der Befehlszeile „ns codename“ ein, wobei „codename“ der Name der Datei ist, den man bei der Code Generierung in NSBM angegeben hat. Ns durchläuft das Simulationsszenario und erstellt das Trace File.
- Um die simulierten Daten zu visualisieren, kann man mit dem Befehl „nam my_output.nam“ (my_output.nam ist das Trace File und wurde von ns erstellt) in der Befehlszeile den Netzwerk Animator starten

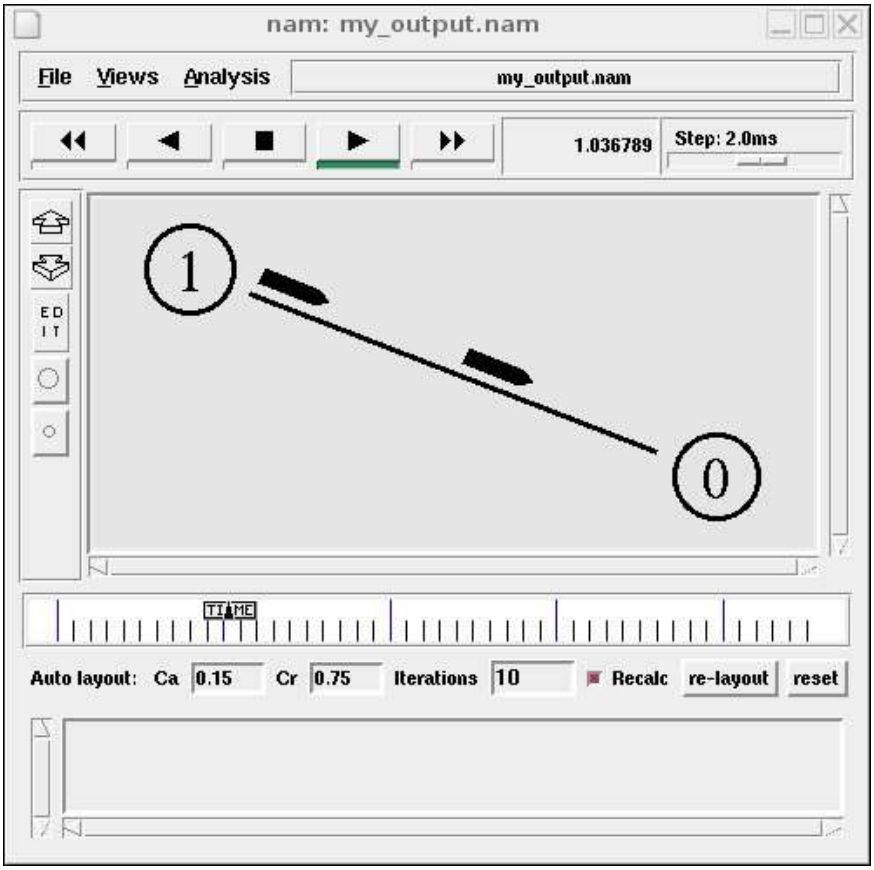

Bild 1: Animierte nam Ausgabe des ersten Beispiels

3.2. Beispiel 2

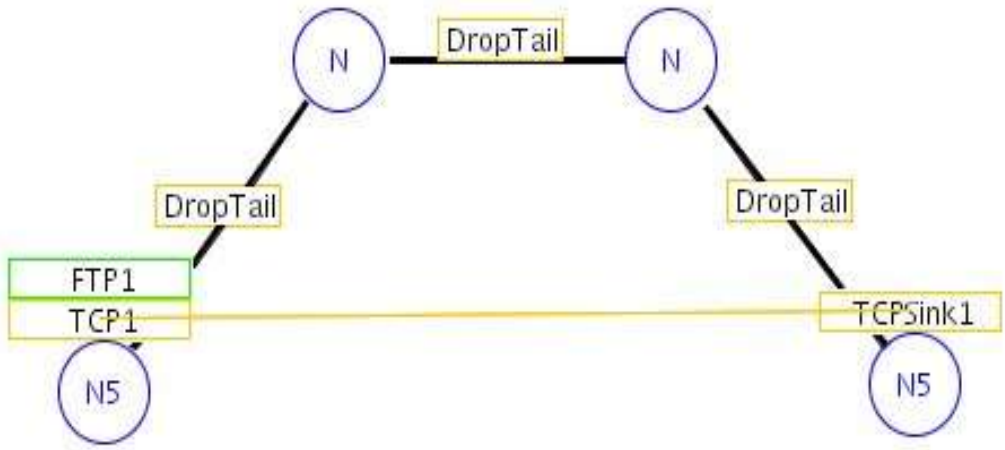

Bild 2: Topologie im NSBM des zweiten Beispiels

Im zweiten Beispiel wird versucht die Funktionalität des NSBM abzudecken. Multiple Knoten sind ebenso vorhanden, wie globale Variablen und Variablen Übergabe direkt beim ns Aufruf in der Kommandozeile. Dieses Mal besteht die Topologie aus vier Knoten, an denen sich an der einen Seite ein TCP Agent und auf der anderen ein TCP Sink befindet; diese sind über die restlichen zwei Knoten verbunden (siehe Bild 2). Die Pakete, die von der FTP Application erstellt werden, werden zum TCP Sink geschickt (d.h. der TCP Agent und das TCP Sink sind miteinander verbunden).
Die neue Funktionalität sind die multiplen Knoten an den Enden der Leitung. Das bedeutet, dass sie in der nam Simulation nicht nur einen Knoten darstellen, sondern mehrere. Wie viele Knoten das sind kann beim Aufruf des ns angegeben werden; diese Anzahl gilt dann für beide Endknoten.

a) Schritte zum Zeichnen der Topologie

Die meisten Schritte zum Zeichnen der Topologie sind sind dem ersten Beispiel ähnlich. Deshalb werden sie hier nicht wiederholt.

- Knoten, Links, Agenten und Applikationen werden wie oben gezeichnet.
- Ein neues Event wird erzeugt (in dem *Timeline* Fenster *FTP1-> New run event*)
- Hinzufügen einer Globalen Variable im „*Variables*“ Fenster (für den Datentyp *Integer* auswählen und mit *add* bestätigen). Anschließend Name und Standardwert eingeben. Beim Markieren der „Commandline“ Option, entscheidet man, dass dieser Wert beim Aufruf des ns angegeben wird.
- Um die multiplen Knoten zu erzeugen, klickt man mit der rechten Maustaste auf den Knoten, den man verändern will und wählt dann *Properties* aus. Im *Properties* Fenster hat jeder Knoten ein *Nodecount*

Attribut, das die Anzahl der Knoten angibt, die der Knoten darstellen soll. Um die globale Variable zu benutzen, markiert man den Knopf neben dem Attribut. Anschließend kann man die bestehenden globalen Variablen auswählen.

- Das Generieren des Codes ist identisch mit dem Beispiel eins.

b) Die Übergabe der Argumente beim Aufruf vom ns

Nachdem der Code generiert wurde, kann ns in mehreren Varianten aufgerufen werden:

- $ ns codename 3
  $ nam my_output.nam
  Dies liefert folgendes Szenario:

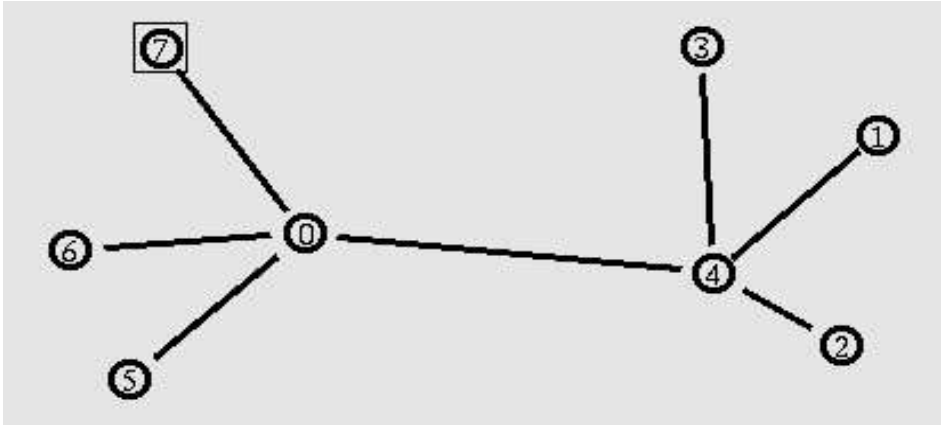

Bild 3: Ergebnis des ns Aufrufs mit dem Parameter 3

- $ ns codename 5
  $ nam my_output.nam
  Dies liefert folgendes Szenario:

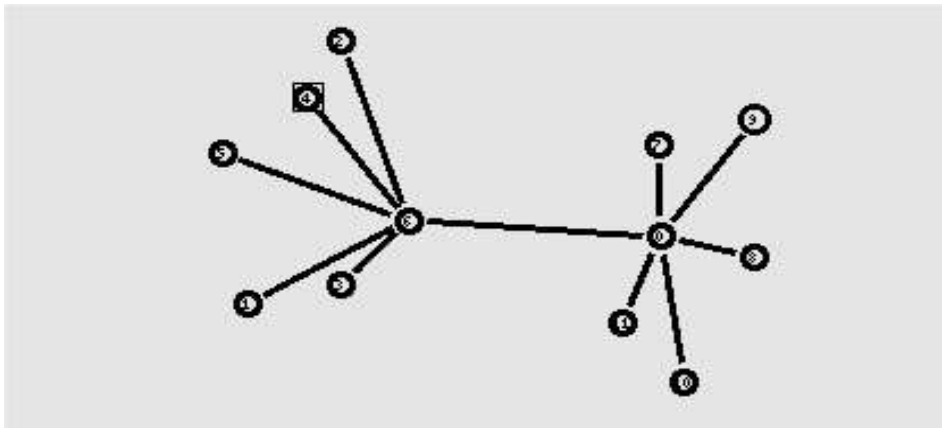

Bild 4: Ergebnis des ns Aufrufs mit dem Parameter 5

4.Die Interna vom NSBM

4.1.Grundstruktur

Die Grundstruktur von NSBM (siehe Bild 5) ist in drei Schichten geteilt, die die wesentlichen Grundfunktionen gliedern. Die erste Schicht ist die XML Configuration Schicht. In ihr wird die Erweiterbarkeit des NSBM gewährleistet. Die zweite Schicht ist die Kernel und GUI Schicht, in der die Funktionalität des Netzwerkzeichnens, die grafische Event-Bearbeitung und die Ansteuerung der Codegenerierung enthalten ist. Als letzte liefert die Code Generator Schicht die Basis, um aus den Informationen der oberen Schichten den OTcl Code zu generieren.
Diese Trennung in drei Schichten bietet die Möglichkeit, alle drei Ebenen unabhängig voneinander zu verändern und erweitern. Diese Aufteilung ist an das Model, View, Controller Pattern angelehnt und bietet auch dessen Vorteile.
In der XML Configuration Schicht werden aus den XML Files über den XML Renderer die Informationen extrahiert, die anschließend für die Erzeugung der ns Objekte (Agenten, Queues, Links und Applications) verwendet werden. Sie beinhaltet alle Spezifikationen der verschiedenen ns Objekte. Um diese Information aus den XML Dateien zu erhalten und sie anschließend in Objekte verwandeln zu können, wurde die „Java Architecture for XML Binding (JAXB)“ verwendet, die später beschrieben wird (siehe Kapitel 4.3). Nach dem Auslesen und Füllen der ns Objekte stehen dem Kernel und GUI Schicht die nötigen Objekte zur Verfügung, um die GUI aufzubauen und nach der Eingabe einer Netzwerktopologie das OTcl Script zu erzeugen.

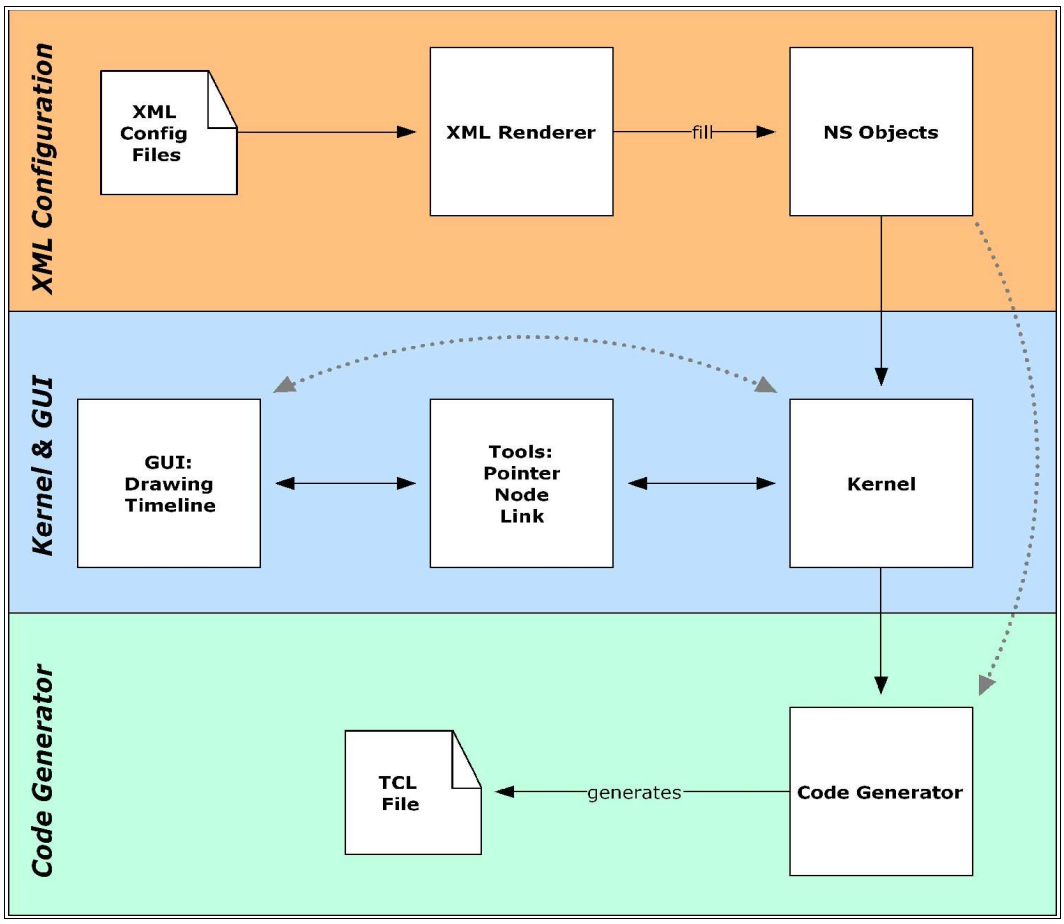

Bild 5: Interner Aufbau des NSBM

Die Kernel und GUI Schicht dient im Wesentlichen dazu, die Funktionalitäten von NSBM dem Benutzer zur Verfügung zu stellen.
Die Schicht, die die Hauptaufgabe von NSBM erfüllt, ist die Code Generator Schicht. Wie oben erwähnt stellt sie den anderen Schichten die Funktionen zur Codegenerierung zur Verfügung. Ihr werden über den Kernel die nötigen Informationen über die zu simulierende Netzwerktopologie übergeben. Mit den zusätzlichen Informationen aus den XML Daten kann nun der Tcl Code erstellt werden.

4.2.Klassendiagramm

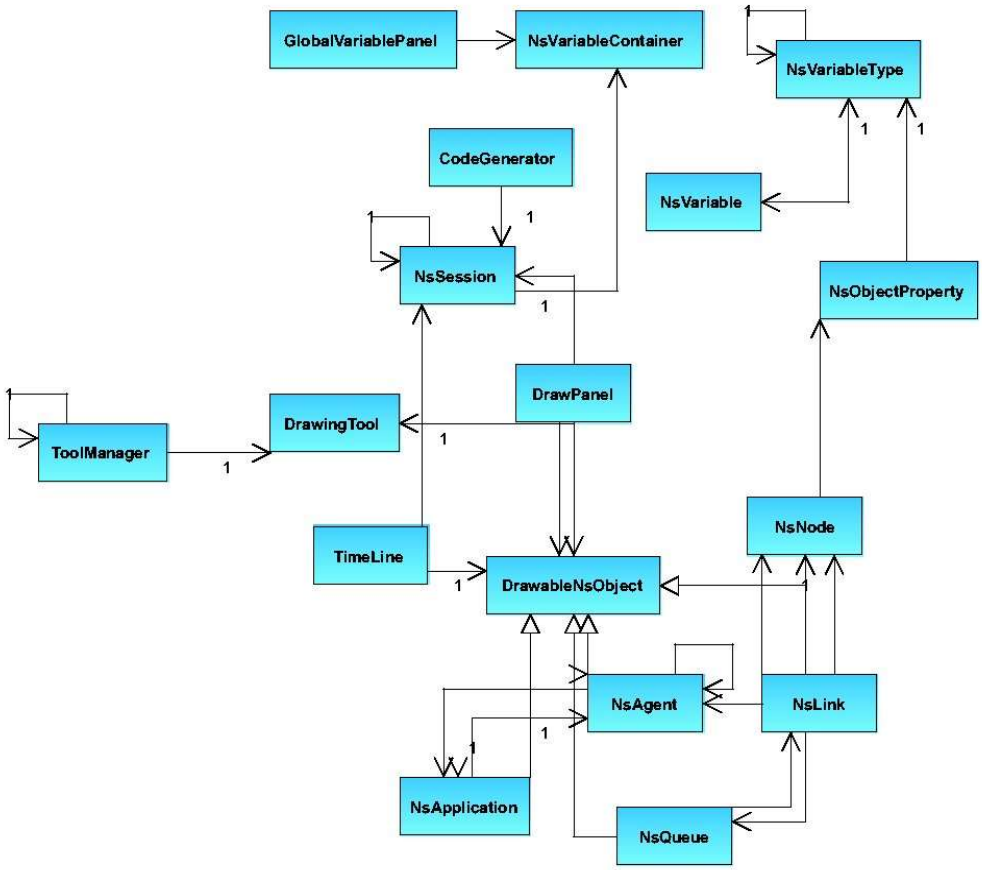

Bild 6: Klasssendiagramm des NSBM

a) GlobalVariablePanel

Diese Klasse bietet dem Benutzer eine grafische Verwaltung der verwendeten Variablen in NSBM – siehe auch NsVariableContainer.

b) NsVariableContainer

Diese Klasse besitzt alle verfügbaren Variablen in einer Session. Hier wird auch festgelegt, ob eine Variable in NSBM definiert wird oder später beim Start von ns über die Kommandozeile angegeben wird.

c) NsVariableType

Diese Klasse stellt eine Wrapper Klasse der elementaren Speichertypen (Integer, Double, usw.) dar. Sie bietet zusätzliche Prüfmöglichkeiten der Inhalte einer Variable - siehe auch NsObjectProperty

d) Codegenerator

Die Klasse Codegenerator Klasse besitzt Grundfunktionen um ein TCL File zu erzeugen. Alle speziellen Codefunktionen liegen in Unterklassen oder werden aus den XML Files bezogen. Um ein TCL File zu erzeugen, benötigt die Klasse lediglich eine Referenz auf das NsSession Objekt, das alle Informationen zu einer Simulations-Session beinhaltet.

e) NsVariable

Diese Klasse speichert den Inhalt einer Variable in NSBM und definiert zusätzlich den Typ der Variable mit Hilfe der NsVariableType Klasse.

f) NsSession

Dies ist die Hauptklasse, die alle Parameter einer Simulation enthält. In dieser Klasse sind alle ns Objekte, ihre Parameter, Events, Einstellungen, usw. gespeichert. Die Klasse wird auch zur Speicherung (Serialisierung) einer Session in eine Datei verwendet.

g) NsObjectProperty

Diese Klasse wird zur Speicherung von verschiedensten Parametern verwendet und definiert weiters, ob ein Parameter statisch definiert wird, oder über eine Variable gesteuert wird. Zur Speicherung wird die Klasse NsVariable bzw. NsVariableType verwendet.

h) DrawPanel

Diese Klasse verwaltet die gesamte Zeichenfläche und alle Interaktionen mit dem Benutzer und stellt daher die Hauptklasse in Bezug auf die Interaktion mit dem Benutzer dar.

i) DrawingTool

Diese Elternklasse aller Tools (Pointer, Node, Link) besitzt die Grundfunktionen eines Tools und schreibt die benötigten Klassen zur Kooperation mit dem DrawPanel den Kindklassen vor.

j) Toolmanager

Diese Klasse verwaltet die möglichen Tools (Pointer, Node, Link), die zur Interaktion mit dem Benutzer verwendet werden.

k) TimeLine

Diese Klasse verwaltet die grafische Zeitleiste und alle darin enthaltenen Events.

l) DrawableNsObject

Diese ist die Basisklasse für alle gezeichneten Objekte (Node,Agent,Applikation..). Sie definiert, grundlegende Schnittstellen zur Ereignisbehandlung und die Interaktion mit dem DrawPanel.

m) NsNode

Diese Klasse beinhaltet alle nötigen Methoden um mit der GUI zu kooperieren und speichert alle Parameter in NsObjectProperty Objekte.

n) NsAgent

Diese Klasse beinhaltet alle nötigen Methoden um mit der GUI zu kooperieren und wird mittels Objekten, die aus XML Files generierten werden, parametrisiert. Die Parameter werden in NsObjectProperty Objekte gespeichert.

o) NsLink

Diese Klasse beinhaltet alle nötigen Methoden um mit der GUI zu kooperieren und wird mittels Objekten, die aus XML Files genriert werden, parametrisiert. Die Parameter werden in NsObjectProperty Objekte gespeichert.

p) NsQueue

Diese Klasse beinhaltet alle nötigen Methoden um mit der GUI zu kooperieren und wird mittels Objekten, die aus XML Files genriert werden, parametrisiert. Die Parameter werden in NsObjectProperty Objekte gespeichert.

q) NsAppliction

Diese Klasse beinhaltet alle nötigen Methoden um mit der GUI zu kooperieren und wird mittels Objekten, die aus XML Files genriert werden, parametrisiert. Die Parameter werden in NsObjectProperty Objekte gespeichert.

4.3.XML Binding

Die bereits erwähnte Java Architecture for XML Binding (JAXB) wird bei NSBM eingesetzt, um die XML Konfigurationsdaten zu verarbeiten. Dies ist bei NSBM nicht einfach möglich, da es viele Funktionen bereitstellen muss, die erst in den Konfigurationsdaten zur Laufzeit spezifiziert werden. Um dies

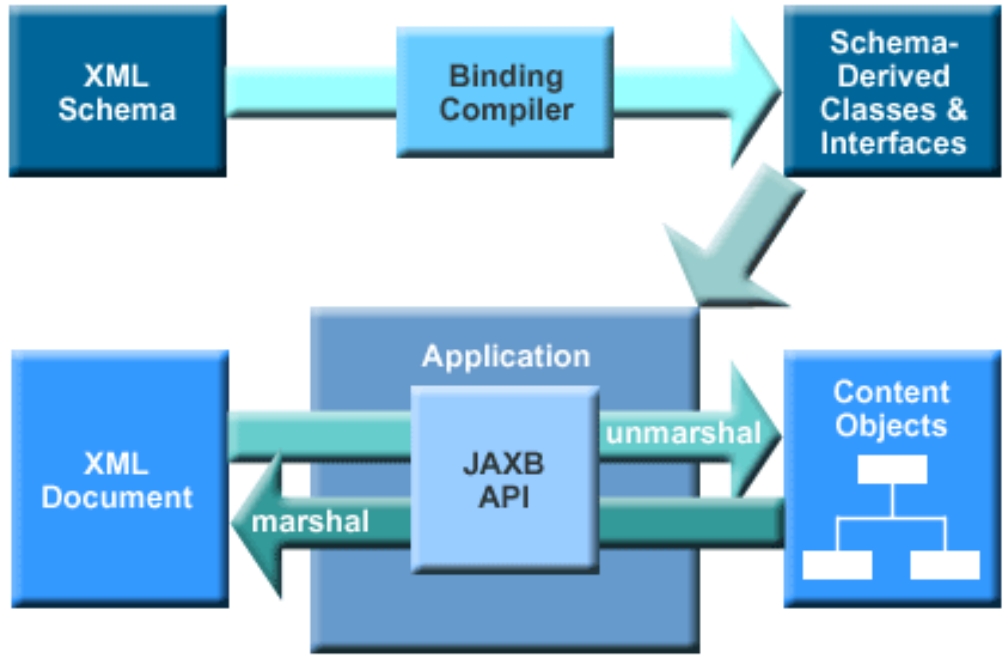

Bild 7: Java Architecture for XML Binding [10]

gewährleisten zu können wurde JAXB eingesetzt. Bei JAXB werden aus den XML Schemata der Konfigurationsdaten bereits bei der Compilierung Klassen und Interfaces erzeugt, die dann zur Laufzeit durch die XML Konfigurationsdaten gefüllt werden können (siehe Bild 7).

4.4.Rendering Engine

a) Gliederung

- Zeichenpanel

  Das Zeichenpanel ist für den eigentlichen Zeichenvorgang und die Ereignisbehandlung verantwortlich. Es delegiert die Mausereignisse bzw. die Zeichenaufrufe an die jeweils zu zeichnenden Objekte weiter.

- Zeichenobjekte

  Zeichenobjekte sind Objekte, die später eine grafische Repräsentation von Netzwerkstrukturen (Nodes,Links,...) widerspiegeln. Sie legen ihre Größe, ihre Position und ihre Zeichenroutine fest.

- Werkzeuge

  Werkzeuge sind Objekte, die zur Erzeugung bzw. Manipulation von Zeichenobjekten dienen.

b) Beschreibung

Das Zeichenpanel leitet sich vom JPanel ab und verwaltet die Zeichen- und Ereignisbehandlung. Darunter ist zu verstehen, dass das Zeichenpanel für das erneute Zeichnen der Grafikobjekte und das Reagieren auf Mausbewegungen und Tastendrücke verantwortlich ist. Diese Ereignisse werden an die Werkzeuge und Zeichenobjekte weitergeleitet. Die Werkzeuge reagieren auf das berühren eines Zeichenobjektes mit der Maus, setzen dieses bei Bedarf als aktiv und ermöglichen dadurch ein Verschieben bei Nodes und ein Umverknüpfen bei Links und Agents. Die Zeichenobjekte hingegen sind rein als Befehlsempfänger anzusehen, die sich bei Bedarf neu zeichnen, auf einfache Signale der Eingabegeräte reagieren (wechseln der Farbe beim Überfahren) und ihre Darstellung verwalten. Zeichenobjekte werden in der jeweiligen Sitzung gespeichert und verwaltet. Sie fungieren damit als Verbindung zwischen Zeichenebene und Programmlogik, da in ihnen auch die Information über ihre jeweiligen Eigenschaften gespeichert ist.

c) Modell

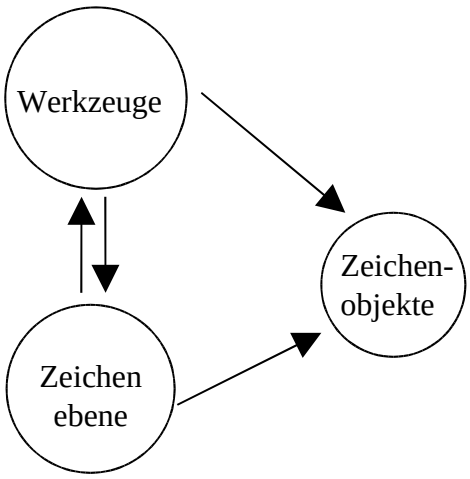

Bild 8: Modell der Rendering Engine

4.5.Framework zur Ereignisverarbeitung

a) Grundidee

Die ersten Implementierungsversuche von NSBM haben sehr schnell die Grenzen der Ereignisbehandlung, wie sie in Java gebräuchlich ist, deutlich gemacht. Dieses Modell ist relativ starr; der Ereignisempfänger benötigt eine Referenz auf die Ereignisquelle um sich als Empfänger registrieren zu können.
Um dieses Problem zu umgehen, wurde ein eigenes schlankes Framework entwickelt, bei dem Ereignisquelle und Ereignisempfänger nur lose miteinander gekoppelt sind. Ereignisse können an mehrere oder auch an gar keine Empfänger gerichtet sein. Dieses Modell ist mit Broadcastmessages in Netzwerken oder einem Bussystem vergleichbar. Jeder Empfänger kann sich für jede Nachricht registrieren. Auch können verschiedene Sender die gleichen Nachrichten aussenden. Eine Nachricht wird rein durch ihre typsichere Signatur definiert.

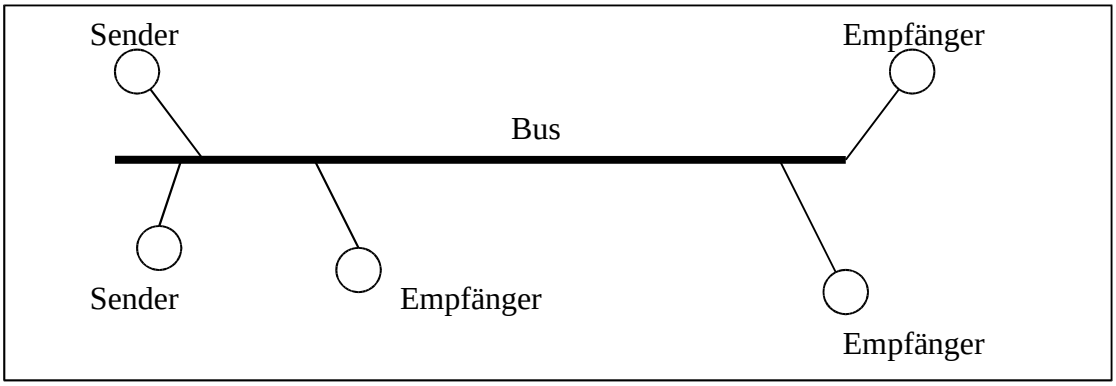

Bild 9: Modell der Ereignisbehandlung

b) Verwendung

Damit ein Objekt Nachrichten empfangen kann, muss es sich zuerst explizit registrieren. Das erfolgt über das Singleton ObjectRegistry mittels folgenden Aufrufes: *ObjectRegistry.getInstance().registerReceiver(this)*. Dabei wird das Objekt nach Methoden, deren Namen auf „Receiver“ enden durchsucht, und diese werden bei Eintreten eines entsprechenden Ereignisses aufgerufen.
Um Ereignisse zu senden, muss eine Klasse das Interface Component implementieren und an einer beliebigen Stelle in ihrer Vererbungshierarchie eine innere Klasse deklarieren, die AbstractEventSender implementiert. Dies Klasse muss dann nur mehr bei der Erzeugung des Objektes instanziert werden und kann in jeden Typ gecastet werden, der als Interface innerhalb dieses Objektes deklariert wurde und vom Interface EventSenderInterface erbt. Durch den Aufruf einer in einem solchen Interface definierten Methode, wird ein Ereignis ausgelöst und an die entsprechenden Empfänger weitergeleitet.

c) Implementierung

Der im letzten Abschnitt erklärte Mechanismus wird hauptsächlich mittels Reflection implementiert. Der Abgleich zwischen Sender und Empfänger wurde dabei typsicher implementiert und erfolgt rein über das Interface der Sender- bzw. Empfängermethode.
Dabei unterscheiden sich die Signaturen der beiden Methoden nur in deren Namen. Der Name des Empfängers entspricht dem Namen des Senders mit einem angehängten „Receiver“. Bei den Sender Objekten handelt es sich um Proxyobjekte, die zur Laufzeit entscheiden welche Interfaces der jeweilige Sender implementiert. Beim Aufruf einer solchen Proxymethode wird ein generischer Handler aufgerufen, der aufgrund der Methodensignatur die Empfängermethoden aus einer Tabelle sucht und diese aufruft.
Diese Art der Implementierung ermöglicht es Sender und Empfänger gänzlich unabhängig voneinander zu verwalten.

d) Anwendungsbeispiele

- Menüpunkte

  Normalerweise benötigt man zur Erzeugung einer Menüstruktur eine Referenz auf das JFrame, in dem diese angezeigt werden soll. Durch die Kapselung der Erzeugung von Menüpunkten in einer von JFrame abgeleiteten Klasse als Nachrichtenempfänger und die Kapselung des Nachrichtensenders in einer Menüpunkt-Klasse, kann an beliebiger Stelle im Code ein Menüpunkt erzeugt bzw. aktiviert werden, ohne dass eine direkte Verknüpfung mit dem JFrame vorhanden sein muss.

- Zeichenobjekte

  Die Zeichenobjekte die in NSBM verwendet werden (Nodes, Links, Agents), verfügen alle über Methoden wie z.B. „repaint“. Die Signalisierung an die eigentliche Zeichenmethode die sich in einem Panel befindet erfolgt auch über den zuvor beschriebenen Mechanismus der Ereignisbehandlung.

- Serialisierung von Zeichenobjekten

  Beim Laden einer gespeicherten Sitzung in NSBM erfolgt eine automatische Registrierung am Event Bus. Diese Objekte sind damit sofort wieder für Ereignisse empfangsbereit.

Generell eignen sich Anwendungen in denen Komponenten nur sehr lose gekoppelt sind für den Einsatz dieses Frameworks, da die unterschiedlichen Komponenten nicht voneinander abhängen und nur bei Bedarf interagieren.

4.6.Code Generator

Der Code Generator ist ein eigenständiger Programmteil des NSBM und übernimmt alle Tätigkeiten um ein TCL File aus dem gezeichnetem Szenario für den ns Simulator zu erzeugen.
Dieser Programmteil kennt grundsätzlich keine Details der jeweiligen NsObjects bzw. wie diese in TCL realisiert werden. Der Code Generator kennt lediglich eine grobe Struktur des TCL Files und stellt zusätzlich noch statisch und dynamisch erzeugte Methoden zur Verfügung.
Die jeweiligen Details, wie ein spezielles NsObject (z.B.: Null Agent) in TCL umgesetzt wird, erhält der Code Generator aus den XML Konfigurations-Dateien. Diese enthalten Informationen wie zum Beispiel TCL Objekt Namen, Dokumentationen und Parameter der jeweiligen Objekte. Aus diesen Informationen kann der Code Generator nun den TCL Code erzeugen.
Um den TCL Quellcode übersichtlich, flexibel und wieder verwendbar zu halten, werden alle Befehle im Quellcode kommentiert.
Da NSBM vor allem für große Netzsimulationen eingesetzt werden soll unterstützt dieser auch multiple Nodes (Anhang A, Kapitel 3, Absatz a „Nodes“). Dennoch muss ein übersichtlicher Quellcode gewährleistet bleiben. Dies wird durch Einzelaufrufe von automatisch erzeugten Methoden pro multiple Node erreicht. Hierzu dienen statische Methoden wie zum Beispiel zur Erzeugung eines multiple Nodes bzw. dynamische Methoden die aus den jeweiligen Informationen aus den XML Dateien zusammengesetzt werden (z.B.: spezieller Agent der auf einen multiple Node aufgesetzt wird). So wird garantiert dass auch ein multiple Node der zum Beispiel 100 Nodes simuliert, mit beinahe gleich viel TCL Code-Zeilen generiert bzw. initialisiert werden kann wie ein Szenario mit nur einzelnen Nodes.
Zur zusätzlichen Übersichtlichkeit wird der erzeugte TCL Code in logische Bereiche, wie zum Beispiel Methoden, Nodes, Agents, Zeitangaben, usw., unterteilt.
Weitere Details siehe Anhang A, Kapitel 4 „Understanding the Code“

4.7.Verwendete Tools

a) Ant

Ant [11] ist ein Build-Werkzeug für die Java-Anwendungsentwicklung, das plattformübergreifend konzipiert ist und als Format für Buildfiles XML verwendet. Ant ist ein Tool, das speziell auf die Erfordernisse des Build-Prozesses in der Java-Umgebung zugeschnitten ist. Es wurde im Rahmen des Jakarta-Projekts der Apache Software Foundation als Open Source entwickel. Bei NSBM wurde es neben dem Build-Prozess auch

bei der Generierung der Javadoc Dateien und bei der jar Paket Erzeugung eingesetzt. Bei vielen Arbeitsschritten ersparte uns Ant viel Zeit, da diese durch Erzeugung von verschiedenen Tasks automatisiert werden konnte.

b) Eclipse

Zur Programmierung von NSBM wurde die Open Source Entwicklungsumgebung Eclipse [12] eingesetzt. Sie ist frei erhältlich und bietet mit einem hervorragenden Editor, Debugger und der Unterstützung für CVS-Systeme alle nötigen Funktionen, die man von einer Java-Entwicklungsumgebung erwartet.

c) Subversion (SVN)

Subversion [13] ist ein Open Source Tool, das für die Versionskontrolle der NSBM Dateien verwendet wurde. SVN ermöglichte es, auf Daten, die auf einem zentralen Server gespeichert sind, zuzugreifen. Jede Änderung, jedes Löschen oder jedes Hinzufügen einer oder mehrerer Dateien bedeutet eine neue "Revision". Bei den neuen Revisions werden nur die geänderten Daten gespeichert. Der Vorteil von SVN ist, dass alle Änderungen in allen Dateien mit gespeichert werden. Das ermöglichte es auch, auf ältere Versionen zuzugreifen.
Durch Subversion wurde die Protokollierung des gesamten Entwicklungsprozesses automatisch übernommen. Auch die Stabilität und besondere Features, wie zum Beispiel die Handhabung von Binärdateien, fielen positiv auf.

d) TortoiseSVN

TortoiseSVN [14] ist ein Open Source SVN Client, der die Handhabung von SVN sehr vereinfacht. TortoiseSVN fügt sich in den Windows Explorer ein und zeigt durch die Symbole im Windows Explorer den Zustand der Ordner bzw. Dateien an. Auch werden Einträge zum Kontextmenü des Windows Explorers hinzugefügt, die das Commiten, Updaten oder andere SVN - Befehle aufrufen.
Durch die Integration in den Windows Explorer lies sich SVN fast nahtlos in die gewohnten Arbeitsabläufe einbinden. Daraus ergab sich ein äußerst kurzer Einarbeitungsaufwand.

e) phpBugTracker

Um Bugs des NSBM zu dokumentieren benutzten wir das Open Source tracking Tool phpBugTracker [15]. So konnten die entdeckten Probleme und Fehler gut, schnell und vor allem zentral protokolliert werden. Somit konnte eine schnelle Fehlerbehebung ermöglicht wurde.

Literaturverzeichnis/Referenzen

[1] Simulations Werkzeuge Andrea Emilio Rizzoli
http://www.idsia.ch/%7Eandrea/sim/simtools.html#network

[2] Opnet [online]
http://www.opnet.com/

[3] REAL network simulator
http://www.cs.cornell.edu/skeshav/real/overview.html

[4] Ns building Seite
http://www.isi.edu/nsnam/ns/ns-build.html

[5] Projekthomepage des ns
http://www.isi.edu/nsnam/ns/

[6] Marc Greis's tutorial [online]
http://www.isi.edu/nsnam/ns/tutorial

[7] Der ns-2 Netzwerksimulator
http://www.welzl.at/research/tools/ns/ns-doku.pdf

[8] The Network Simulator – ns-2
http://www.isi.edu/nsnam/ns/ns-documentation.html

[9] ns tool Seite
http://welzl.at/research/tools/ns/

[10] Java Architecture for XML Binding
http://java.sun.com/developer/technicalArticles/WebServices/jaxb/

[11] Apache Ant 1.6.5 Manual [online]
http://ant.apache.org/manual

[12] Eclipse [online]
http://www.eclipse.org/

[13] Version Control with Subversion [online]
http://svnbook.red-bean.com/

[14] TortoiseSVN [online]
http://tortoisesvn.tigris.org/docs.html#DocDir
[15] phpBugTracker
http://phpbt.sourceforge.net/docs/
[16] Network Animator – nam
http://www.isi.edu/nsnam/nam
[17] TCLWise – Guide to the programming language [online]
http://www.invece.org/tclwise/
[18] LEO Wörterbuch [online]
http://dict.leo.org/

A Anhang

1.Introduction

First of all, NSBM is the name of a program and secondly, the acronym for Network Simulator by Mouse. This shows that NSBM has to do something with ns sometimes also called ns-2. If you wonder what ns or ns-2 is and hope that this will be explained here, you are wrong. An explanation about ns-2 already exists on http://www.isi.edu/nsnam/ns/. Here you also find other links to get started. Some German beginners documentation is on http://www.welzl.at/research/tools/ns.

Before NSBM was built, the only way to use ns-2 was to write tcl code to describe nodes, links and other network objects. Now, as the name NSBM suggests, you can save some work by using your mouse to generate the tcl code. In fact you can draw a network topology with multiple nodes with only a few mouse clicks. Afterwards you click on a button and there is the tcl code, almost ready for use with the ns. Why almost? NSBM can do a lot of things: multiple nodes, fast changes in a comprehensible visual way, common variables and other things, but usually you have to advance the code to the topology you had in mind. There is a section (chapter 4.) on understanding the generated code but I suggest to read the Marc Greis tutorial (http://www.isi.edu/nsnam/ns/tutorial) before you start with NSBM.

If you already have experience with drawing tools, and you have read the Greis tutorial or know the basic concepts of network simulations with ns, you should start with the chapter “Installation and launch“. After that, you can skip chapter 3, “The graphical user interface“, or read it if you are interested in something in particular. The next step is to understand the code, this is explained in chapter 4. And only after this, if you are interested to expand the function of code generation, you should read the last chapter “Code generation and the xml files”. It explains how the program generates the tcl code. In fact there are some xml files which specify what names are used in the code.

2.Installation and launch

For a full enjoyment of NSBM you need at least a java runtime environment (version 1.4.*, don't try the 1.5.* version). Of course you also need ns itself. You can find it at http://www.isi.edu/nsnam/ns/ns-build.html; we made good experiences with the all-in-one-package.
You can find NSBM at http://welzl.at/research/tools/ns. To launch NSBM you have to type “java -jar ns-20041222.jar” either in a console or the command line. The numbers behind “ns-” are the building date (maybe you have a newer version). You have to pay attention that all executable paths for the java VM are set.

3.The graphical user interface

In this chapter you get information about all the things appearing in the GUI (Graphical User Interface), on how they work, and how you can build the network topology, that you want to analyze.

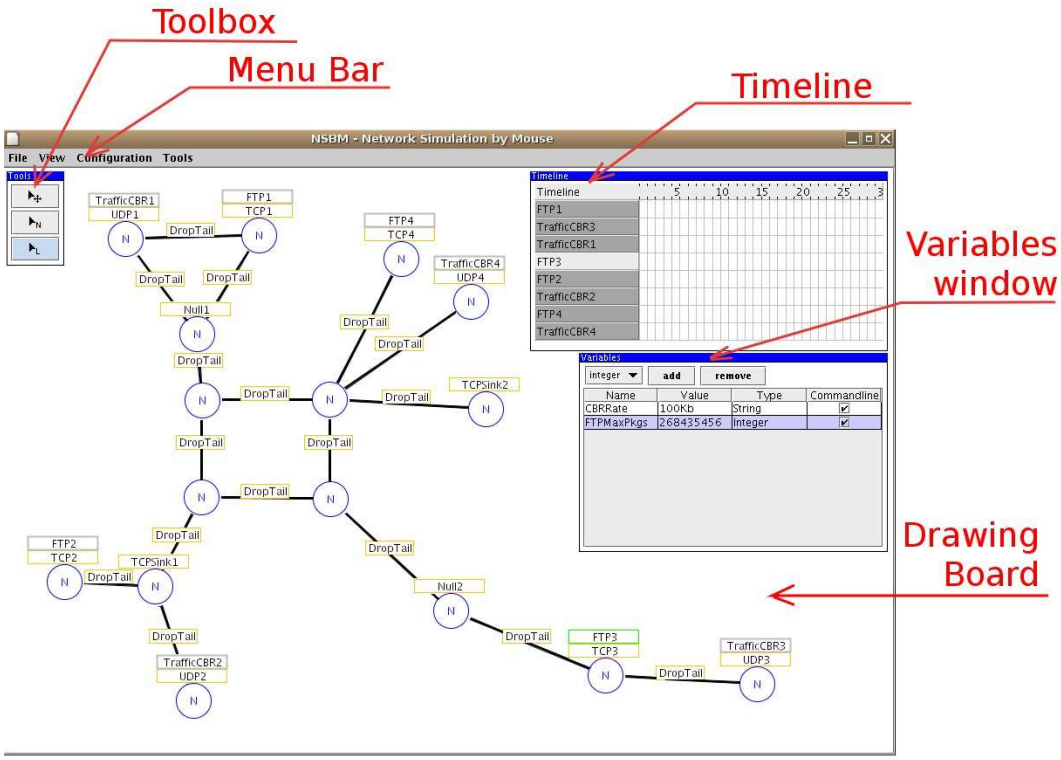

Figure 1: windows and boxes with an example

The toolbox:

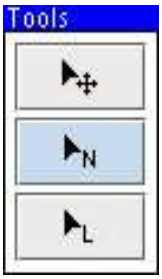

Figure 2: the tool box

As in common drawing programs, you have a tool box where you can find all the things you need for drawing, in our case you will need them to draw a network topology.

a)Node

Figure 3: the node tool

As you know, every computer, router or other network device in ns is a node. It is rather simple to use the node tool: you select it by clicking on its symbol and afterwards you can add nodes on the board (see Figure 1) wherever you want. This new generated nodes have a lot in common: their properties are all set to a standard value (specified in the configuration xml files; explained in the chapter about code generation), and they have no connections to other links yet.

You can change the attributes by pointing with the mouse on the circle that represents a node and right clicking. A menu will appear where you can remove nodes, add agents (see chapter 3.1, “Adding Agents and Applications“), and change the properties of nodes.

A node has two properties: it has a name which appears in the generated code and a “Nodecount”. This “Nodecount” is a bit trickier than the other things we discussed until now:

Many network topologies have a similar structure: there is a node (for example an Internet provider), and this node is linked to other nodes which all have the same properties (in the real world they may be the customers of the Internet provider).

You might think that this is no problem: you have a fast right hand, a fast index finger and a new mouse, and drawing a lot of nodes is easy. You are right. But if you have to change properties on nodes or want to seek one of these nodes in the generated code you will be very happy with the

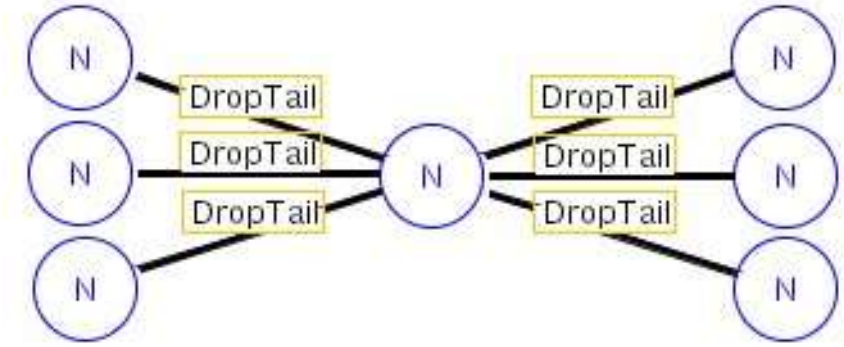

Figure 4: a dumbbell topology

Nodecount feature:

the “Nodecount” allows you to represent many nodes with identical properties in one single node. This means that if you have the Internet provider mentioned above (which is a single node), and you want to collect the customers (for ex. 3) in one node you have to change the Nodecount to 3, and that's it. If you change properties of the customer's node this change automatically occurs in every single node. The same thing happens with the links between the customer's and the provider's node.

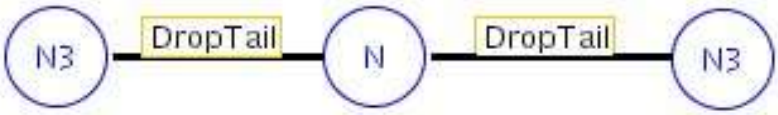

Figure 5: the dumbbell topology from Figure 4 with the Nodecount feature

b)Link

Figure 6: the link tool

In the explanation above, you have seen two figures with nodes and multiple nodes. If you blindly followed this documentation so far, you would not be able to draw the mentioned examples because the links between the nodes are missing.

Drawing links it is not complicated at all. You only have to keep in mind that if you have multiple nodes you have also multiple links (a look at the Figure 4 and Figure 5 makes this clear).

Let us look at the properties of the links: There are two different property windows. If you right click on the black line (the link itself) you get the properties of the link (name, link type, queue type, delay and bandwidth). I won't explain the details of every field because much depends on the configuration xml's. But a small hint: if you shortly remain on the input field with your mouse you will get help as on the picture below!

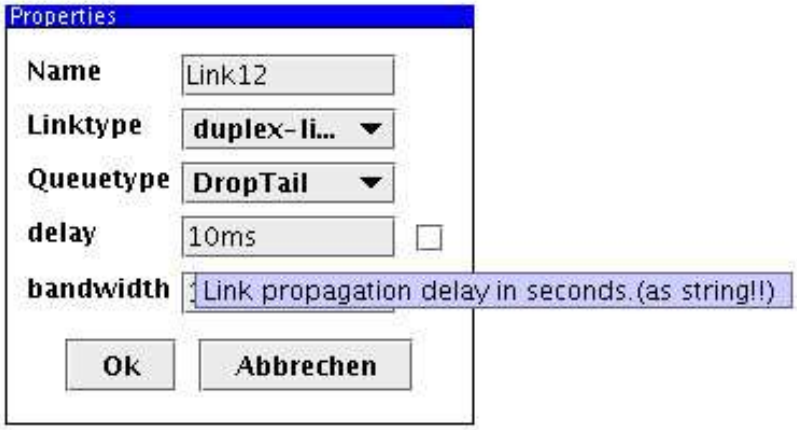

Figure 7: link properties and the help function

If you right click on the queue type label (the label surrounded by the yellow box) you get the properties of the queue. If you are already wondering about the little box on the right side of the properties you have to be a little patient; it will be explained in chapter 3.2, “Variables“.

c)Move

Figure 8: the move tool

Finally, the simplest tool: the move tool. With this, you can move things around to create a nice looking topology ;). Note that changing the position of nodes does not affect the generated code. Also node that the right button of the mouse is still working! You can always change properties or remove things while you reorganize your “picture”.

3.1.Adding Agents and Applications

As you surely know, the TCP protocol (in the OSI model) is located at the transport layer. If we speak about agents we are therefore talking about the transport layer. You can specify this layer for every node by right clicking on it and selecting “Add Agent”. Then you have to select the Agent that you want. Which agents are available is specified in the configuration xml's.

The next step is the application layer. As the name suggests, this is where applications are located. It is the same process as with the agents: right click on the agents label (the yellow box), select “Add applications” and then select the application you want. Now you may notice that you see a part of the OSI model (the yellow and the green boxes).

The last thing you have to bear in mind is that you should indicate the point of departure and arrival of every package. Usually it is defined in the network layer, but since there is no network layer in ns it has to be defined between two nodes (or more nodes if multiple nodes are affected). With the link tool, point to the departure agent and draw a line to the arrival agent.

3.2.Variables

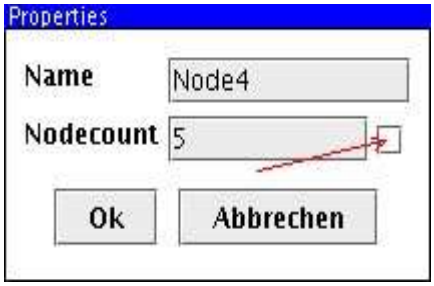

Figure 9: the properties window with the variables check box

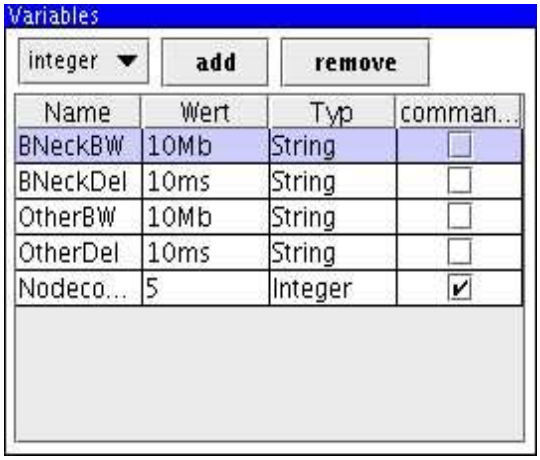

Figure 10: global variables window

Now we come to the mysterious little box on the right side of some of the properties (look at Figure 9 above). As you can see in the figure (look at Figure 10), there are some variables in this window (as is explained later). You have to imagine all the variables as global variables, and you can use them in your topology by selecting the afore mentioned little box. In the generated code, these are in fact global variables. Afterwards you can select the right variable from the drop down list near the “magic” little box. With this feature, for instance, you're able to change a lot of links and bandwidths with a few clicks (think of the afore mentioned dumbbell topology).

How can you generate global variables in the GUI, you might ask? Rather simple: select the appropriate data type from the drop down list in the variables window (Figure 7). Be aware, some properties accept only certain strings; bandwidth, for instance, only accepts strings like “10Mb”. Use the help function in the property windows (see Figure 7) to find out which data type to use. Then click on the “add“ button to create the new variable. Now the variable is ready to get changed by clicking on the value field and typing

in the new value.

If you have a topology and you want to test it with different variables rather than generate new code you can specify that you pass the values over the command line. To specify this, you have to check the box “commandline”. In the example 2 (you can find it in the documentation package), for instance, you have to start ns with one argument: “ns codename 100Kb”.

Perhaps you have realized what happened after clicking on the little box in the properties window? The field on the left side of the box now says “default”.

3.3.Time line

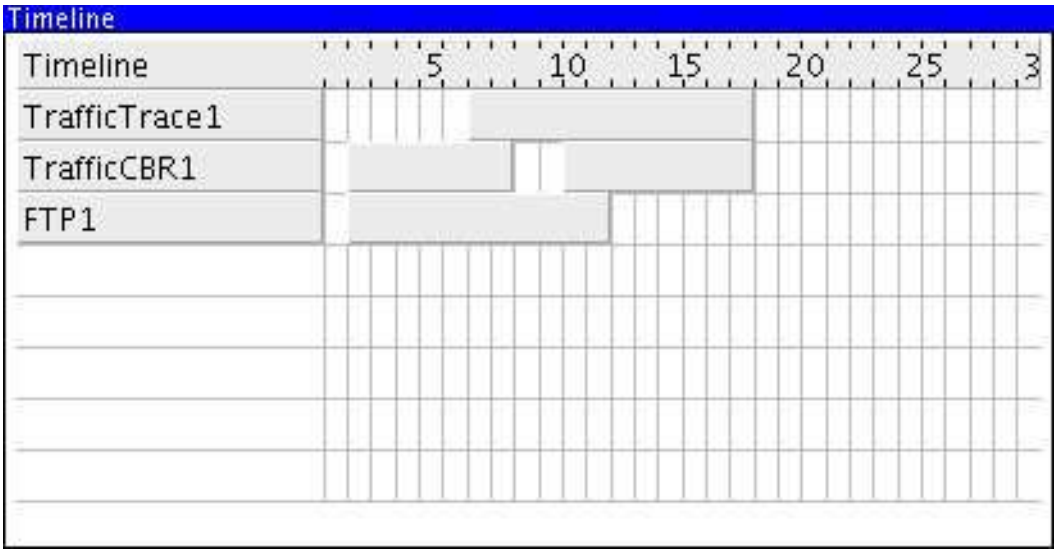

Figure 11: the time line window

In the time line window you can control the sending behavior of the applications. This means that you can control at which point an application starts and stops generating packets. The action of starting and stopping an application is called event. Such events can always be added by right clicking on an application label (or on the application labels in the topology itself). Note, that you can add more than one event per application (as you can see in the Figure 11 above).

The tricky thing in this window is the time grid. In the standard configuration every intercept corresponds to one second. But if this is changed (see chapter 3.6, “Menu Configuration“), things get more complicated. This means that if one intercept corresponds to ½ second, the time at the intercept 10 will be 5 (10 multiplied with ½). If you have some problems with this, compare the example1.nsm (located in the dictionary of the documentation) and compare it with the output.

3.4.Menu Files

Among the buttons “New”, “Load...”, “Save...” and “Close” (the meaning should be self explanatory) you can also find the button “Generate NS Code”. This button does what the entire program was created for: it generates the tcl code for ns depending on the drawn network topology and the configuration (we will discuss this in chapter 3.6, “Menu Configuration“).

Figure 12: the files sub menu

3.5.Menu View

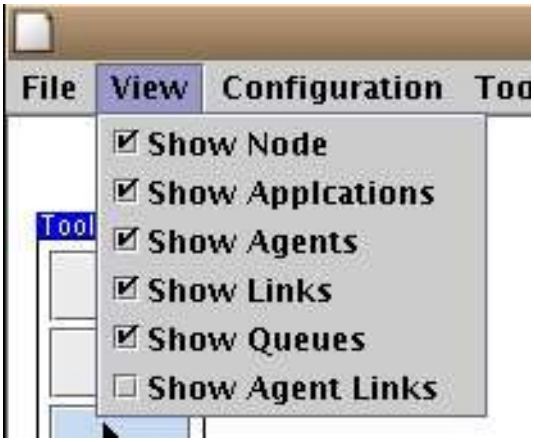

Figure 13: the view sub menu

All items in the view menu are selected:

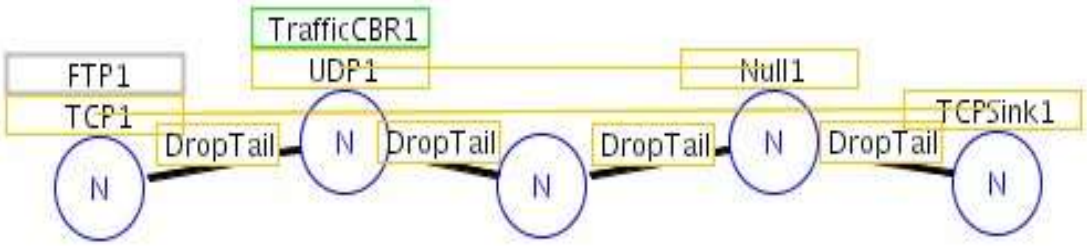

Figure 14: all views selected

Only the check box “Show Node” is selected:

Figure 15: only nodes selected

Only the check box “Show Applications” is selected:

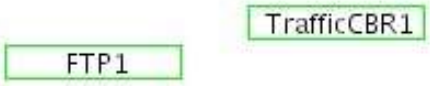

Figure 16: only applications selected

Only the check box “Show Agents” is selected:

Figure 17: only applications selected

Only the check box “Show Links” is selected:

Figure 18: only links selected

Only the check box “Show Queues” is selected:

Figure 19: only queues selected

Only the check box “Show Agent Links” is selected:

Figure 20: only agent links selected

When reading this chapter, you may have realized that there are two concepts of linking: the link between two nodes, and links between agents. This link between agents defines from where to where packets should travel.

3.6.Menu Configuration

Figure 21: the configuration sub menu

In the menu “Configuration” you will only find the entry “Session configuration”- but behind this entry are a lot of configurations. If you click on it, a configuration window appears. The function of this window can be divided in two groups: the name and type of the output file that ns produces, and the time line options.

a) name and type of ns output

There are two types of output files which can be generated from ns: the nam file, the trace file or both. By marking the check boxes with the self explanatory titles on the left you can select which one is to be generated. As you can see, you can also change the file names. These are simply the names for the generated files in your file system.

b) time line options

These options are already mentioned in the chapter “Time line“. Here the important thing that you have to keep in mind is that the labels on the time line are not always seconds but more generally time units. The “Runlength” indicates not the time but the number of total intercepts. For example, if the time unit is ½ and the “Runlength” is 10, the simulated time is 5 seconds (10 multiplied by ½).

3.7.Menu Tools

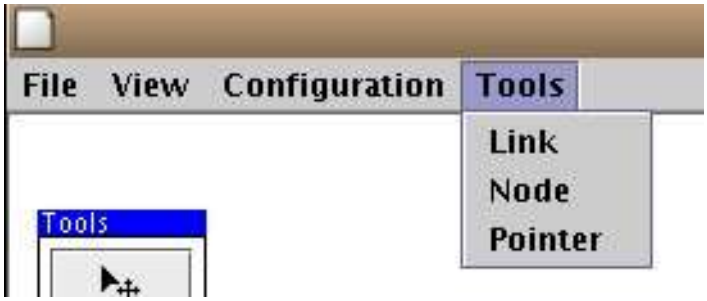

Figure 22: the tools sub menu

In the Tool Menu you can find the same options as in the Tools window.

4.Understanding the code

This chapter tries to explain what code structures are generate by NSBM and how you can change them into a running code. In the generated code you can find a lot of tcl procedures for multiple nodes, multiple links and other complex tcl code. But don't worry. We begin with a simple example in order to understand what NSBM can generate.

4.1.Code Overview

The generated code is split up in to several parts. All parts are separated by comments describing the following code. These comments begin with “#” and are contained in every generated code.

Here is a rough overview of the code structure. If you read through this code you see numbered parts, starting without “#”. In these sections you find an explanation about the contained code. Usually, you will find the tcl code for ns in the generated code. The numbers in front of every explanation indicate the section numbers for reference purposes below.

```
#############################################
# Tcl Script created by NSBM - Network Simulation by Mouse
# Copyright 2004
#############################################
1. on this place is the cration of a new simulator object and the finish proc
#############################################
#
# Procs to manage multiple nodes
#
#############################################
2. A lot of procs for multiple nodes
#############################################

#################### Global Variables
3. Global variables
#################### Nodes Code
4. creation of nodes objects
#################### Links Code
5. all about links
#################### Queues Code
6. all about queues
#################### Agents Code
7. all about agents
#################### Applications Code
8. the applications stuff
#################### Time Events
9. all the time events and the
"$ns run" statement
```

4.2.Simple code parts

Let us now start with the code which was generated from the example1.nsm. It is contained in the documentation package and you can generate it by yourself with NSBM by loading the example file and clicking on “Generate NS Code” in the File menu. The result of your code generation should be like the file “codeExample1.tcl” in the documentation package. Let us go through the important code parts:

Section one:

```
set ns [new Simulator]

set nf [open my_output.nam w]
$ns namtrace-all $nf

proc finish {} {
  global ns nf tf
  $ns flush-trace

  close $nf
  #exec nam my_output.nam &
  exit 0
}
```

In this code section all code lines are best explained in the Greis tutorial.

In section two, many procedures are used to create “multiple nodes”

In this code section all code lines are best explained in the Greis tutorial.

In section two, many procedures are used to create “multiple nodes” topologies. Since we have no multiple nodes in the first example, we can ignore these procedures of section two. However the code is always generated.

The same in section three: no global variables in example1.nsm => no code there.

Section four:

```
#################### Nodes Code
set Node1 [$ns node]
set Node2 [$ns node]
```

Here two new node objects are created (again, see the Greis tutorial for details).

Section five:

```
#################### Links Code
$ns duplex-link $Node1 $Node2 1Mb 10ms DropTail
set Link1 [$ns link $Node1 $Node2]
```

The first code line is explained in the Greis tutorial and the second line assigns a handle (named Link1) to the link.

Section six:

```
#################### Queues Code
set DropTail1 [$Link1 queue]
 $DropTail1 set unblock_on_resume_ true; #dokuin
 $DropTail1 set limit_ 50; #dokuin
 $DropTail1 set blocked_ false; #dokuin
```

Here a new queue object is created and some properties are set up.

Section seven:

```
#################### Agents Code
set Null1 [new Agent/Null]; #null
set UDP1 [new Agent/UDP]; #null
 $UDP1 set packetSize_ 1000.0; #
$ns attach-agent $Node2 $Null1
$ns attach-agent $Node1 $UDP1
$ns connect $Null1 $UDP1
```

See Greis tutorial.

Section eight:

```
#################### Applications Code
set TrafficCBR1 [new Application/Traffic/CBR]; #null
 $TrafficCBR1 set random_ 0; #
 $TrafficCBR1 set interval_ 0.0050; #
 $TrafficCBR1 set rate_ 448Kb; #
 $TrafficCBR1 set packetSize_ 210.0; #
 $TrafficCBR1 set maxpkts_ 268435456; #
$TrafficCBR1 attach-agent $UDP1
```

A new CBR application is created and its properties are set up. In the last

```
#################### Time Events
$ns at 0.5 "$TrafficCBR1 start"
$ns at 4.5 "$TrafficCBR1 stop"
$ns at 5.0 "finish"
$ns run
```

At last, there are the time events in the generated code.

4.3.Advanced code parts

In the example2.nsm file, we introduce two new functions of NSBM: a global variable and in at the same time a command line variable. Obviously, the code changes in the Global Variables section.

```
#################### Global Variables
set Interval [lindex $argv 0]
```

Here, a global variable is set via the command line (argv 0). The ns command is the following: “ns codename 0.006”, where the codename is the name of the generated tcl file and 0.006 is the interval value.

In the example3.nsm file, things are more complex:

Here we have an example with the Nodecount feature which was already mentioned in chapter 3, “The graphical user interface“ What has changed in the generated code?

The procedures in the code section:

```
# Procs to manage multiple nodes
```

are used. All these procedures have a similar structure: the parameters are set and afterwards, the nodes, links, agents, applications or properties of them are generated, linked, attached, connected or set in a loop. For example, a single node is generated via the code line `set Node1 [$ns node]` but multiple nodes need the procedure “createMultipleNode”, where you have to pass as parameters a node array and the number nodes to create.

```
proc createMultipleNode {nodes number} {
      upvar $nodes n
      global ns
      for {set i 0} { $i < $number} {incr i} {
            set n($i) [$ns node]
      }
}
```

The names of these procedures are self explanatory, and if there are many parameters they are described in the code.

5.Code generation and the xml files

The xml files are a very important part of NSBM. All information about applications, agents, queues, links, their properties, and their compatibility are stored there. In order to extend the functionality of NSBM, you can simply change the xml files and you practicaly end up with a new application of your choice. There are four xml files in NSBM: agent.xml, application.xml, links.xml and queues.xml.

Every one of the items in the xml files includes information about itself and information about its compatible items.

Note that the xml files depend on each other and you have to control this interaction by yourself. The dependencies are as follows:

- There is a <possibleAgents> tag in the agents.xml which defines what agents can be connected. Obviously, the possible Agent must also be defined in the agents.xml.
- If agents.xml contains a possible application, it has to be defined in the applications.xml.
- In links.xml contains possible queues, they declared have to be also in the queues.xml

In the next chapters the structure of the xml files will be shown. You can add more tags where you see three points (...). Most of the tags should be understandable by their name.

5.1.Agents

<agent> one or more different agents are defined in agents.xml. A new agent description starts with this tag.

<configObject> in this tag, all attributes concerning the agent are located.

<name> this is the name in the menus of NSBM.

<codename> this string is used when the code is generated.

<description> is the help you get if you leave the mouse pointer over an agents property.

<documentation> is for the code documentation. In the generated code you will find the string located in this tag on the right of the concerning code part.

<properties> all properties that an agent can have are located here.

<property> a single property.

<type> the type of the value (string, integer, double,...).

<default> a default value that is used when an agent is created.

<possibleAgents> here all possibly connected agents are listed. All possible agents also have to be defined in the agents.xml.

<possibleApplications> here, all applications that can be attached to the agent are listed. All possible applications also have to be defined in application.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<agents xmlns="http://ns/configuration/generator/">
      <agent>
            <configObject>
                  <name>UDP</name>
                  <codename>UDP</codename>
                  <description>a basic UDP agent</description>
                  <documentation>not documented yet</documentation>
                  <properties>
                        <property>
                              <name>Packet size</name>
                              <type>double</type>
                              <default>1000</default>
                              <description>The maximum segment...
                              </description>
                              <codename>packetSize_</codename>
                              <documentation></documentation>
                        </property>
                        <property>
                        </property>
                        ...
                  </properties>
            </configObject>
            <possibleAgents>
                  <possibility>Null</possibility>
                  <possibility></possibility>
                  ...
            </possibleAgents>
            <possibleApplications>
                  <possibility>Traffic/Exponential</possibility>
                  <possibility></possibility>
                  ...
            </possibleApplications>
      </agent>
      <agent>
      </agent>
      ...
</agents>
```

5.2. Applications

<application> enclosed in this tag, all attributes of an application are defined.

<name> this is the name in the menus of NSBM.

<codename> this string is used when the code is generated.

<description> is the help you get if you leave the mouse pointer over an agents property.

<documentation> is for the code documentation. In the generated code you will find the string located in this tag on the right of the concerning code part.

<possibleMethods> and <method> here, all possible methods are defined; at the moment NSBM does not use these possibilities.

<property> a single property.

<type> the type of the value (string, integer, double,...).

<default> a default value that is used when an agent is created.

```
<?xml version="1.0" encoding="UTF-8"?>
<applications xmlns="http://ns/configuration/generator/">
      <application>
            <name>Traffic/CBR</name>
            <codename>Traffic/CBR</codename>
            <description>CBR_Traffic generates traffic according ...
            </description>
            <documentation></documentation>
            <possibleMethods>
                  <method>start</method>
                  <method></method>
                  ...
            </possibleMethods>
            <properties>
                  <property>
                        <name>Interval</name>
                        <type>double</type>
                        <default>0.005</default>
                        <description>(Optional)interval between ...</description>
                        <codename>interval_</codename>
                        <documentation></documentation>
                  </property>
                  <property>
                  </property>
                  ...
            </properties>
      </application>
      <application>
      </application>
      ...
</applications>
```

5.3.Links

<link> a new definition of a link with all attributes and possibilities starts with this tag.

<name> this is the name in the menus of NSBM.

<codename> this string is used when the code is generated.

<description> is the help you get if you leave the mouse pointer over an

agents property.

<documentation> is for the code documentation. In the generated code you will find the string located in this tag on the right of the concerning code part.

<properties> a list of single properties concerning the link.

<property> a single property.

<type> the type of the value (string, integer, double,...).

<default> a default value that is used when a agent is created.

<possibleQueues> here, different possibilities are listed (see <possibilities>).

<possibilities> a single possible queue. The queue has to be included in the queues.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<links xmlns="http://ns/configuration/generator/">
      <link>
            <configObject>
                  <name>duplex-link</name>
                  <codename>duplex-link</codename>
                  <description>This creates a bi-directional ...
                  </description>
                  <documentation></documentation>
                  <properties>
                        <property>
                              <name>bandwidth</name>
                              <type>string</type>
                              <default>1Mb</default>
                              <description>Link bandwidth in bits per second...
                              <codename>bandwidth_</codename>
                              <documentation>simple the bandwidth</documentation>
                        </property>
                        <property>
                        </property>
                        ...
                  </properties>
            </configObject>
            <possibleQueues>
                  <possibility>DropTail</possibility>
                  <possibility></possibility>
                  ...
            </possibleQueues>
      </link>
      ...
</links>
```

5.4.Queues

<queue> a single queue is described after this tag.

<name> this is the name in the menus of NSBM.

<codename> this string is used when the code is generated.

<description> is the help you get if you leave the mouse pointer over an agents property.

<documentation> is for the code documentation. In the generated code you will find the string located in this tag on the right of the concerning code part.

<possibleMethods> and <method> here, all possible methods are defined; at the moment NSBM does not use these possibilities.

<properties> a list of different properties.

<property> a single property.

<type> the type of the value (string, integer, double,...).

<default> a default value that is used when a agent is created.

```
<?xml version="1.0" encoding="UTF-8"?>
<queues xmlns="http://ns/configuration/generator/">
  <queue>
    <name>DropTail</name>
    <codename>DropTail</codename>
    <description>test</description>
    <documentation>doku</documentation>
    <possibleMethods/>
    <properties>
      <property>
        <name>Limit</name>
        <type>integer</type>
        <default>50</default>
        <description>The queue size in packets</description>
        <codename>limit_</codename>
        <documentation>dokuin</documentation>
      </property>
      <property>
      </property>
      ...
    </properties>
  </queue>
  <queue>
  </queue>
  ...
</queues>
```