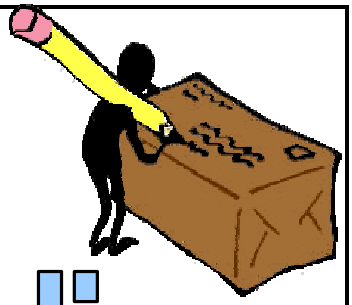
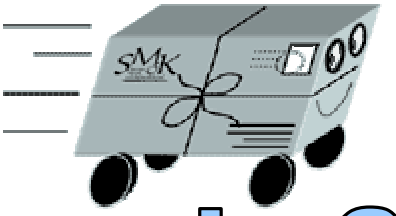
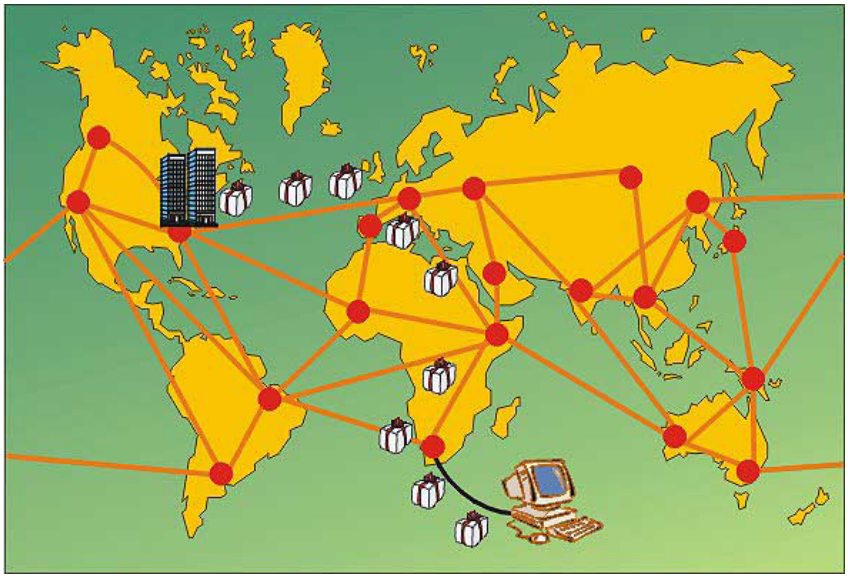


VeriNeC Studio



ein GUI für den



VeriNeC Simulator

Eine Bachelorarbeit von Renato Löffel.
(renato.loeffel@unifr.ch)

In Zusammenarbeit mit
Dominik Jungo und *David Buchmann*.

Gruppe TNS , Universität Fribourg, 2004
(diuf.unifr.ch/tns/verinec)

1 Einleitung

Verinec ist ein Informatikprojekt der Universität Freiburg und steht für *Verified Network Configuration*. Es handelt sich dabei um einen Netzwerksimulator, welcher es erlauben soll, Netzwerke auszutesten, bevor diese physikalisch realisiert werden. Absicht ist es, jene Netzwerke auf deren Sicherheit sowie deren Funktionstüchtigkeit zu prüfen. Um dieses Vorhaben umzusetzen, werden Tests definiert, die auf das Netzwerk angewandt werden. Sobald eine Netzwerkkonfiguration sämtliche Tests besteht, soll diese exportiert werden und zwar in ein Hersteller-unabhängiges Format, sodass sie problemlos auf Netzwerkkomponenten verschiedenster Anbieter eingespielt werden können.

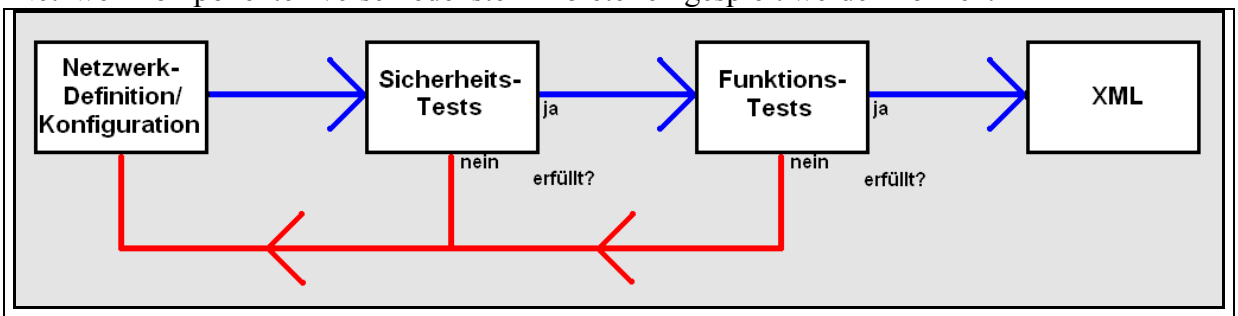


Abbildung 1: Verinec Ablauf

2 Die graphische Oberfläche

Die graphische Oberfläche stellt nur einen Teil des *Verinec* Projektes dar. Sie basiert auf dem eigentlichen *Simulator*, welcher als separates Unterprojekt geführt wird und indirekt vom *Translator*. Die graphische Oberfläche verfolgt zwei Absichten. Einerseits stellt sie Mechanismen für die Netzwerkdefinition zur Verfügung, andererseits visualisiert sie die Testabläufe auf jenen Netzwerken. Dementsprechend besitzt das Programm zwei Modi, genannt Definitionsmodus resp. Simulationsmodus.

2.1 Voraussetzungen

Bevor das Projekt ausgeführt werden kann, muss sicher gestellt werden, dass alle Dateien und Programme auf dem jeweiligen Computer verfügbar sind. Hier eine Übersicht:

• Java Runtime Environment	v.1.4.2_03 oder höher	java.sun.com
• Jdom.jar	version b-10 oder höher	www.jdom.org
• junit.jar	archiv File	www.junit.org
• desmoj.jar	archiv File von Framework	www.desmoj.org
• netsim.jar	archiv File vom Simulator	diuf.unifr.ch/tns/verinec
• netman.jar	archiv File von Translator	diuf.unifr.ch/tns/verinec
• gui.jar	archiv File vom Gui	diuf.unifr.ch/tns/verinec

2.2 Der Definitionsmodus

Vor der eigentlichen Simulation muss das zuerst Netzwerk definiert werden.. Es können Knoten (i.e. Computer, Firewalls, Routers, Switches, etc.), Schnittstellen, sowie Verbindungsleitungen zwischen jenen Schnittstellen erstellt werden. Bei den Schnittstellen handelt es sich um Netzwerkkarten wie z.B. Ethernet, Wirelesskarten oder Modemschnittstellen. Die jeweiligen Verbindungsleitungen entsprechen dem Typus der Schnittstelle. Eine Leitung zwischen zwei Ethernetschnittstellen erscheint als grüne durchgezogene Linie, während eine Wirelessverbindung blau und gestrichelt dargestellt wird.

2.2.1 Anwendung

Nachdem das Programm aufgestartet wird, erscheint auf dem Display ein Fenster, welches jenem aus *Abb. 2* ähnlich sieht. Im Wesentlichen besitzt das Fenster eine Menuleiste, drei Werkzeugleisten, sowie eine Seitenspalte. Beim Aufstarten befindet sich das Programm im „Definitionsmodus“.

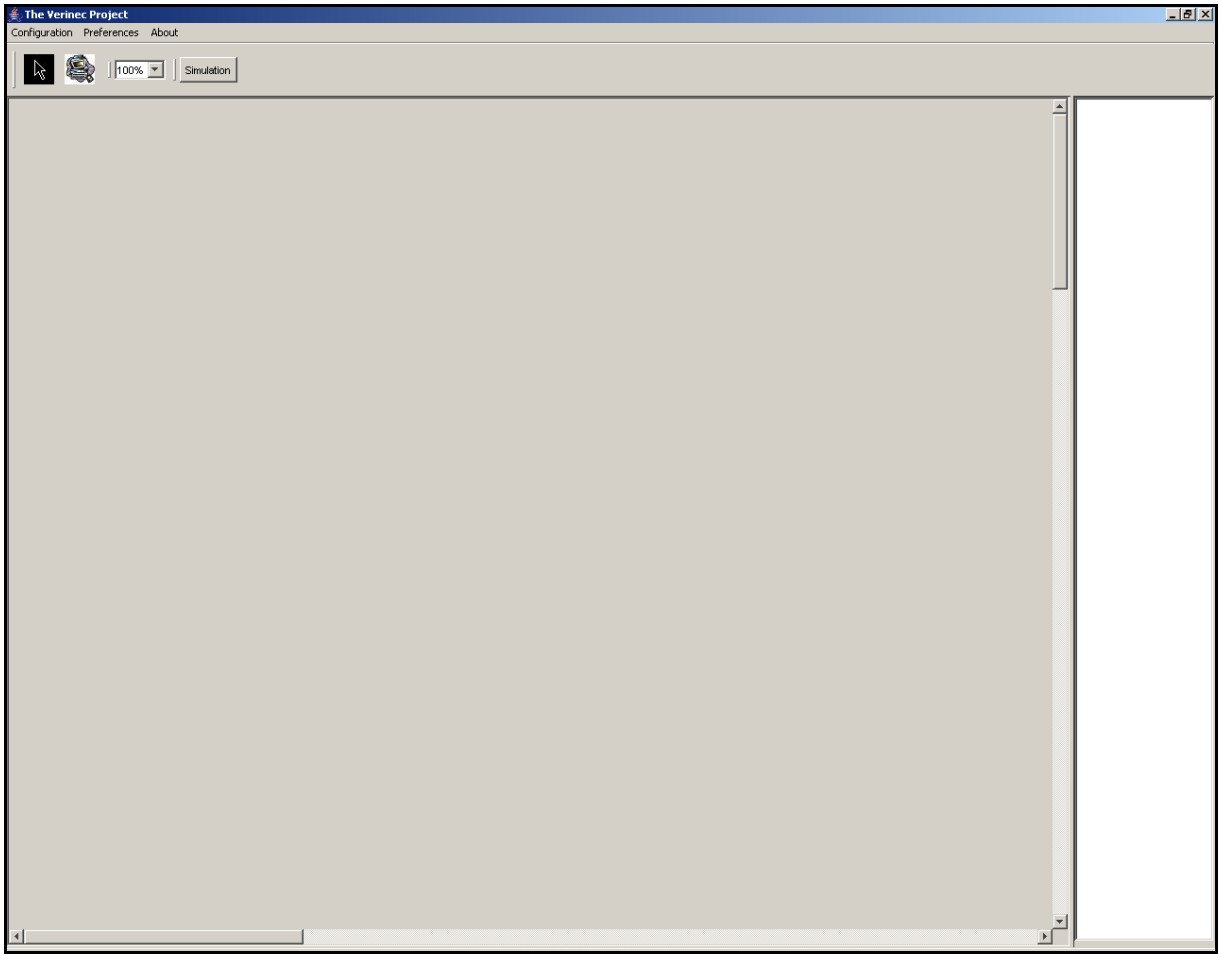


Abbildung 2: Startfenster

2.2.1.1 Die Menuleiste

Innerhalb der Menuleiste finden sich drei Einträge, wobei nur der Erste (*Configuration*) von Bedeutung ist. *Preferences* wurde bis zum jetzigen Zeitpunkt noch nicht implementiert, und *About* gibt lediglich Informationen bezüglich des Projektes an. Sobald der Menupunkt *Configuration* angeklickt wird, entfaltet sich ein Untermenu welches in *Abb. 3* ersichtlich ist. Der User kann anhand des Untermenus entweder eine neue Netzwerkkonfiguration erstellen (*New*), eine bestehende laden (*Load*), oder die aktuelle Konfiguration abspeichern (*Save, Save As ...).*

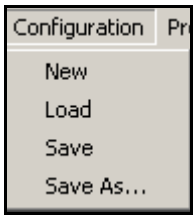


Abbildung 3: Menu

2.2.1.2 Die Werkzeugleisten

In der Toolbar befinden sich drei Werkzeugleisten, welche in *Abb. 4* zu sehen sind. Die Erste dient zum Erstellen der Knoten. Dabei kann mit der Maus entweder der Pfeil, oder der Knoten ausgewählt werden. Welcher Effekt die jeweilige Auswahl erzielt, wird später diskutiert.
Mit Hilfe der zweiten Werkzeugleiste, kann die Arbeitsfläche vergrössert resp. verkleinert werden. Dies ist besonders nützlich, falls das Netzwerk grosse Dimensionen annimmt und eine komplette Übersicht erforderlich wird.
Bei der dritten Werkzeugleiste handelt es sich lediglich um einen Knopf. Dieser Knopf dient dazu, den Modus der Applikation zu wechseln. Beim Drücken des Knopfes im Definitionsmodus, wechselt die Applikation in den Simulationsmodus und vice-versa.

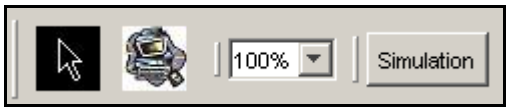


Abbildung 4: Werkzeugleiste

2.2.1.3 Das Seitenfenster

Das Seitenfenster (*Abb. 2 rechts*) gibt im Definitionsmodus Auskunft über den derzeitig angewählten Knoten. Dabei wird die XML-Definition des entsprechenden Knotens angezeigt. Diese Information dient lediglich als Kontrolle für den User. Demnach ist es nicht möglich, den Inhalt jenes Seitenfensters zu modifizieren.

2.2.1.4 Die Erstellfläche

Die Erstellfläche (*Abb. 2 links*) ist das Herz des Definitionsmodus. Auf dieser Fläche können sämtliche Netzwerkkomponenten erstellt werden. Wie die jeweiligen Komponenten kreiert werden, soll nun demonstriert werden.

2.2.1.4.1 Knoten

Ein Knoten repräsentiert ein Gerät mit Netzwerkfunktionalität, wie zum Beispiel Switches, Routers, Firewalls, Computers, etc. Ein derartiger Knoten kann wie folgt kreiert werden:
Als erstes muss in der Werkzeugliste das Knotenicon ausgewählt werden. Sobald nun der Mauszeiger auf die Erstellfläche gezogen wird, erscheint ein gestricheltes Quadrat, welches als Vorschau dient (*Abb. 5*). Befindet sich nun diese Vorschau auf der gewünschten Stelle, kann der Knoten durch Betätigung der linken Maustaste erstellt werden.

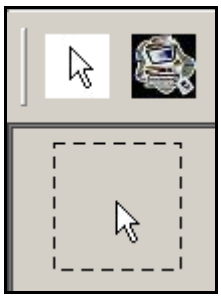


Abbildung 5: Vorschau

Auf der Erstellfläche ist nun der Knoten im angewählten Zustand zu sehen. Im Seitenfenster erscheint sofort die Knotendefinition. Dabei handelt es sich um eine minimale Konfiguration, welche durch das entsprechende Knoten-Schema-File definiert ist.
Es verseht sich von selbst, dass weitere Knoten auf analoge Weise erstellt werden können, wodurch ein ganzes Knotennetzwerk entsteht. Sollte einer der Knoten sich nicht an der gewünschten Stelle befinden, kann dieser durch ein Dragging der linken Maustaste verschoben werden. Sollte nicht nur ein einzelner Knoten verschoben werden, sondern eine ganze Knotengruppe, kann dies mit Hilfe einer Gruppenauswahl geschehen. Dazu müssen die Knoten mit einem Auswahlrechteck eingefangen werden. (*Abb. 6*). Dies wird erreicht, indem auf der Erstellfläche ein Dragging der linken Maustaste durchgeführt wird. Beim Loslassen der Taste, werden nun sämtliche Knoten innerhalb des Rechtecks markiert. Nun kann einer der ausgewählten Knoten verschoben werden, wodurch sich die ganze Knotengruppe verschiebt.

Abbildung 6: Gruppenauswahl

Sobald alle gewünschten Knoten erstellt sind, kann in der Werkzeugleiste der Pfeil wieder angewählt werden, was übrigens auch mit einem Rechtsklick auf der Erstellfläche direkt realisiert werden kann. Bei einem solchen Rechtsklick, werden zudem sämtliche Knotenmarkierungen aufgehoben.
Der Knoten selbst besitzt ein Menu (*Abb. 7*), welches durch einen Rechtsklick aufgerufen werden kann. Das Menu enthält 3 Einträge zum Kreieren einer Schnittstelle (*Add Ethernet, Add Wireless, Add Modem*), einen Eintrag *Delete* zum Löschen des Knotens, sowie der Eintrag *Properties*, welcher dazu dient, das Icon des Knotens zu ändern. Wird *Properties* angewählt, öffnet sich ein Datei-Browser. Es kann nun eine Graphikdatei geladen werden. Wichtig ist, dass es sich dabei um eine Graphikdatei handelt, die von Java™ unterstützt wird.

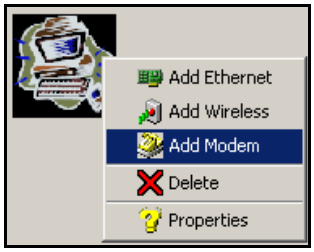


Abbildung 7: Menu

Falls *Delete* ausgewählt wird, wird nicht nur der aktuelle Knoten gelöscht, sondern auch sämtliche markierten Knoten. Da *Delete* eine unwiderrufliche Aktion ist und da keine Rückfrage geschieht, ist Vorsicht geboten.
Falls einer der *Add-Einträge* aufgerufen wird, wird die entsprechende Schnittstelle erzeugt. Vergleiche dazu den folgenden Abschnitt.

2.2.1.4.2 Schnittstellen und Bindings

Die Schnittstellen ermöglichen die Kommunikation zwischen den verschiedenen Knoten. Jede Schnittstelle verfügt über einen gewissen Netzwerktyp und kann nur mit Schnittstellen des selben Netzwerktyps verbunden werden. Diese Applikation unterstützt derzeit die drei Typen Ethernet, Wireless und Serial.
Um eine Schnittstelle zu erzeugen, muss im Menu des gewünschten Knotens einer der drei Add-Einträge (*Add Ethernet, Add Wireless, Add Modem*), gewählt werden. Mit der Auswahl einer dieser Einträgen, wird eine Schnittstelle auf dem Knoten erstellt. Dabei erscheint eine kleine Graphik, welche sich neben den Knoten heftet (*Abb. 8*).
Auffallend ist, dass sowohl die Wireless- sowie auch die Serial-Schnittstelle zwei Graphiken anlegt, während bei der Ethernetschnittstelle lediglich eine Graphik erscheint. Dies kommt daher, dass die Ethernetschnittstelle nur über ein einziges Binding verfügen kann, während die beiden Anderen aus mehreren Bindings bestehen können.

Abbildung 8: Schnittstellen

Ein solches Binding stellt eine logische Verbindung zu einem Netzwerk dar. Dies kann man sich wie folgt vorstellen: Beispielsweise kann ein Knoten der über eine einzige Wirelesskarte verfügt, gleichzeitig in mehreren Netzwerken teilnehmen. Um dies darzustellen, werden diese Bindings verwendet. Schnittstellen mit mehreren Bindings verwenden die selbe Graphik für Schnittstellen sowie für Bindings, wobei jene Graphik, welche direkt den Knoten berührt ,die Schnittstelle darstellt, während die in Kette darauffolgenden Graphiken Bindings repräsentieren. Diese Differenzierung ist wichtig, da die Menus von den Schnittstellen und den Bindings unterschiedlich sind (*Abb. 9*).

Abbildung 9: Schnittstellen und Binding Menus

Die beiden Menueinträge *Delete* und *Properties* sind bei den Schnittstellen sowie bei den Bindings identisch. Mit *Properties* kann das Icon der Schnittstelle, resp. des Binding geändert werden. *Delete* löscht das jeweilige Binding resp. die Schnittstelle (inklusiv der dazugehörigen Bindings). Der Menupunkt *Add Binding* kommt nur bei den Schnittstellen vor. Er dient dazu, ein weiteres Binding anzufügen. Das Resultat ist in *Abb. 10* ersichtlich. Es kann eine beliebige Anzahl von Bindings hinzugefügt werden, jedoch dürften in der Praxis selten mehr als zwei bis drei vorkommen. Der Menupunkt *Deconnect* eines Bindings dient dazu die jeweilige Verbindungslinie zu dem Netzwerk zu trennen. Näheres dazu findet sich im dem nächsten Abschnitt.

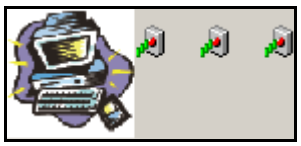


Abbildung 10: Bindings

Ein Knoten kann auch über mehrere Schnittstellen verfügen, welche natürlich nicht unbedingt vom selben Netzwerktypus sein müssen. Falls ein Knoten über eine grosse Anzahl von Schnittstellen verfügt, wird dieser entsprechend vergrössert (*Abb. 11*). Beim Entfernen von Schnittstellen, reduziert sich die Grösse des Knotens wieder auf seine minimale Grösse.

Abbildung 11: Viele Interfaces

2.2.1.4.3 Verbindungslinien und Hubs

Die Verbindungslinien verknüpfen die Knoten (d.h. deren Bindings) untereinander. Sie stellen das physikalische Netzwerk dar. Verbindungslinien weisen ebenfalls einen Netzwerktypus auf, welcher mit dem Typus des Bindings korrespondiert. Die Darstellung der Linien ist dementsprechend verschieden, sodass eine optische Unterscheidung möglich ist. Ethernetverbindungen sind grüne durchgezogenen Linien, Wirelessverbindungen erscheinen als blaugestrichelt und Serialverbindungen zeigen sich in einem mattem Gelb als durchgezogene Linie (*Abb. 12*). Eine solche Linie kann erstellt werden, indem mit der linken Maustaste ein Binding angeklickt wird, wobei die Taste gedrückt bleibt, und auf ein anderes Binding des selben Typus gezogen wird, gefolgt vom Loslassen der Taste. Nun erscheint die Linie entsprechend des Typus des Bindings. Auffallend ist, dass in der Mitte der Linie ein fetter Punkt in der Farbe der Linie erscheint. Bei diesem Punkt handelt es sich um einen Hub, welcher das logische Netzwerk repräsentiert. Dieser Hub kann angewählt werden und im Seitenfenster erscheint seine Definition. Er besitzt ebenfalls ein Menu mit den beiden Einträgen *Delete* und *Properties*, wobei *Delete* den Hub sowie sämtliche Verbindungslinien löscht, welche in ihn münden. Mit *Properties* ist wiederum das Ändern des Icons möglich. Der Hub kann wie ein Knoten auf der Erstellfläche verschoben werden, wodurch die Verbindungslinien ebenfalls eine Transformation erfahren.

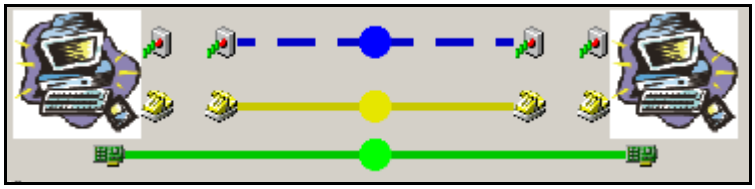


Abbildung 12: Verbindungslinien

Soll nun ein weiterer Knoten mit einem bestehenden Netzwerk verbunden werden, muss wie folgt vorgegangen werden. Mit der linken Maustaste muss wiederum das gewünschte Binding angeklickt werden, wobei die Taste nicht losgelassen wird. Anschliessend muss die Maus auf den Hub gezogen und die Maustaste muss losgelassen werden. Nun ist die gewünschte Verbindung zu dem Netzwerk hergestellt (*Abb. 13*). Wichtig ist, dass nicht auf ein Binding, welches bereits im Netzwerk teilnimmt gezogen wird, ansonsten wird ein neues Netzwerk kreiert, und die Verbindung des Zielbindings zum ursprünglichen Netzwerk wird durchtrennt (*Abb. 14*). Dies wäre in gar keinem Fall eine ungültige Operation, jedoch ist sie in diesem Fall nicht wünschenswert, da die Knoten in unserem Fall allesamt im selben Netzwerk sein sollen.

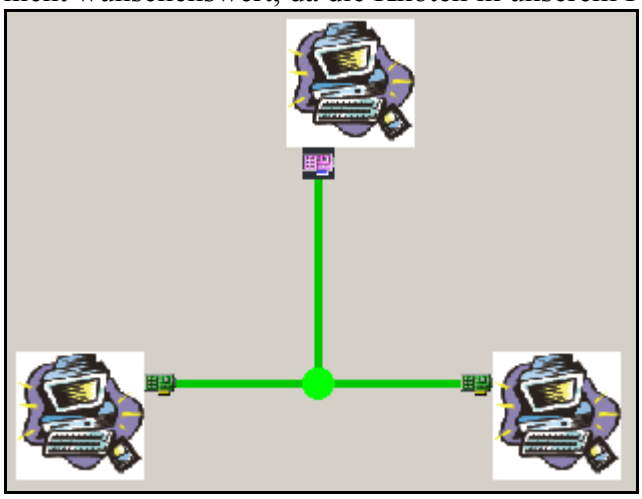


Abbildung 13: ein Netzwerk

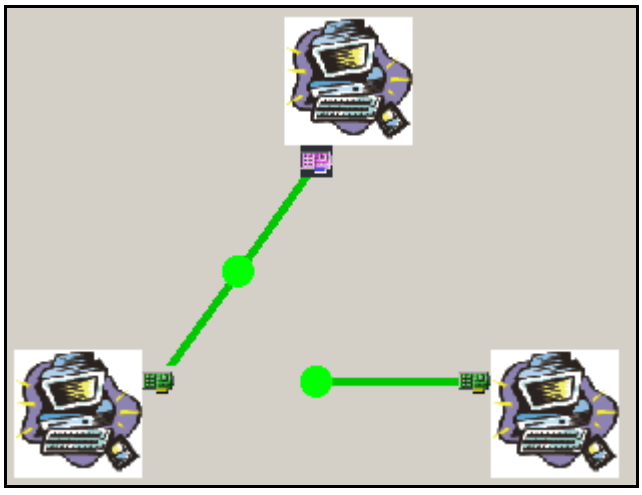


Abbildung 14: zwei Netzwerke

2.2.2 Architektur

In diesem Abschnitt wird beschrieben wie der Definitionsmodus aufgebaut ist. Dazu wird zuerst eine Übersicht über das Package sowie seine Klassen gegeben. Anschliessend folgt ein Kapitel über den Aufbau des Moduls.

2.2.2.1 Das Package und seine Klassen

Das zum Definitionsmodus dazugehörige Package heisst *Configuration* und befindet sich im Ordner *src/Configuration*. Es umfasst 20 Klassen (*Abb. 15*), welche sämtliche Funktionalitäten der Netzwerkdefinition realisieren. Ein Grossteil der Klassen repräsentieren Knoten, Hubs, Schnittstellen, Bindings oder Verbindungslinien. Daneben gibt es aber auch eine Klasse (*NodesToolbar*), welche die Knotentoolbar verkörpert. Zudem gibt es diverse abstrakte Klassen, welche allesamt mit *Nw* beginnen (Beispielsweise *NwComponent*, welches die Hauptklasse darstellt). Wie bereits weiter oben angesprochen, sind nur die drei Netzwerktypen *Ethernet*, *Wireless* und *Serial* implementiert. Dementsprechend gibt es nur Klassen für diese drei Typen, wobei jene Klassen jeweils mit den obigen Wörtern beginnen (*PPP* entspricht *Serial*). Zudem fällt auf, dass zu jeden Netzwerktypus vier Klassen existieren. Diese vier Klassen verkörpern jeweils die Schnittstelle, das Binding, die Verbindungslinie und den Hub des entsprechenden Netzwerktypus. Falls ein weiterer Netzwerktypus hinzugefügt werden soll (siehe Erweiterungsmöglichkeiten), müssen die vier dazugehörigen Klassen erstellt werden.

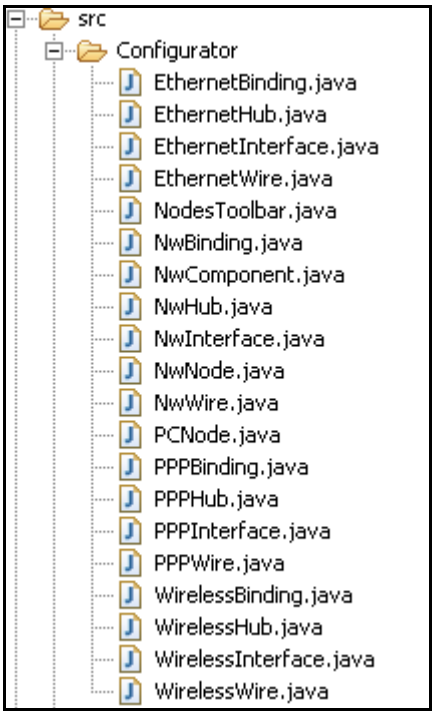


Abbildung 15: Klassen

2.2.2.2 Klassen: Hierarchie und Zusammenspiel

Im Configuratorpackage gibt es eine wichtige Hierarchie, welche nun beschrieben werden soll (*Abb. 16*). An der Spitze der Hierarchie findet sich die Klasse *NwComponent*, welche die Oberklasse für sämtliche Netzwerkkomponenten (*NwBinding, NwInterface, NwNode*) darstellt. Sie beinhaltet sämtliche Felder und Methoden, welche ihre Unterklassen besitzen müssen. Einige Aufgaben werden in der Klasse direkt realisiert, dazu zählen: Koordinaten des GUI an die Knotendefinitionsdatei anzufügen, gewisse Ereignisse zu behandeln und die Grösse resp. die Markierung der Komponenten zu regeln. Zudem besitzt die Klasse ein wichtiges Feld, nämlich die Konfigurationsdatei der Komponente. Diese Konfigurationsdatei charakterisiert die entsprechende Komponente vollständig und wird jeweils bei der Markierung der Komponente im Seitenfenster angezeigt.

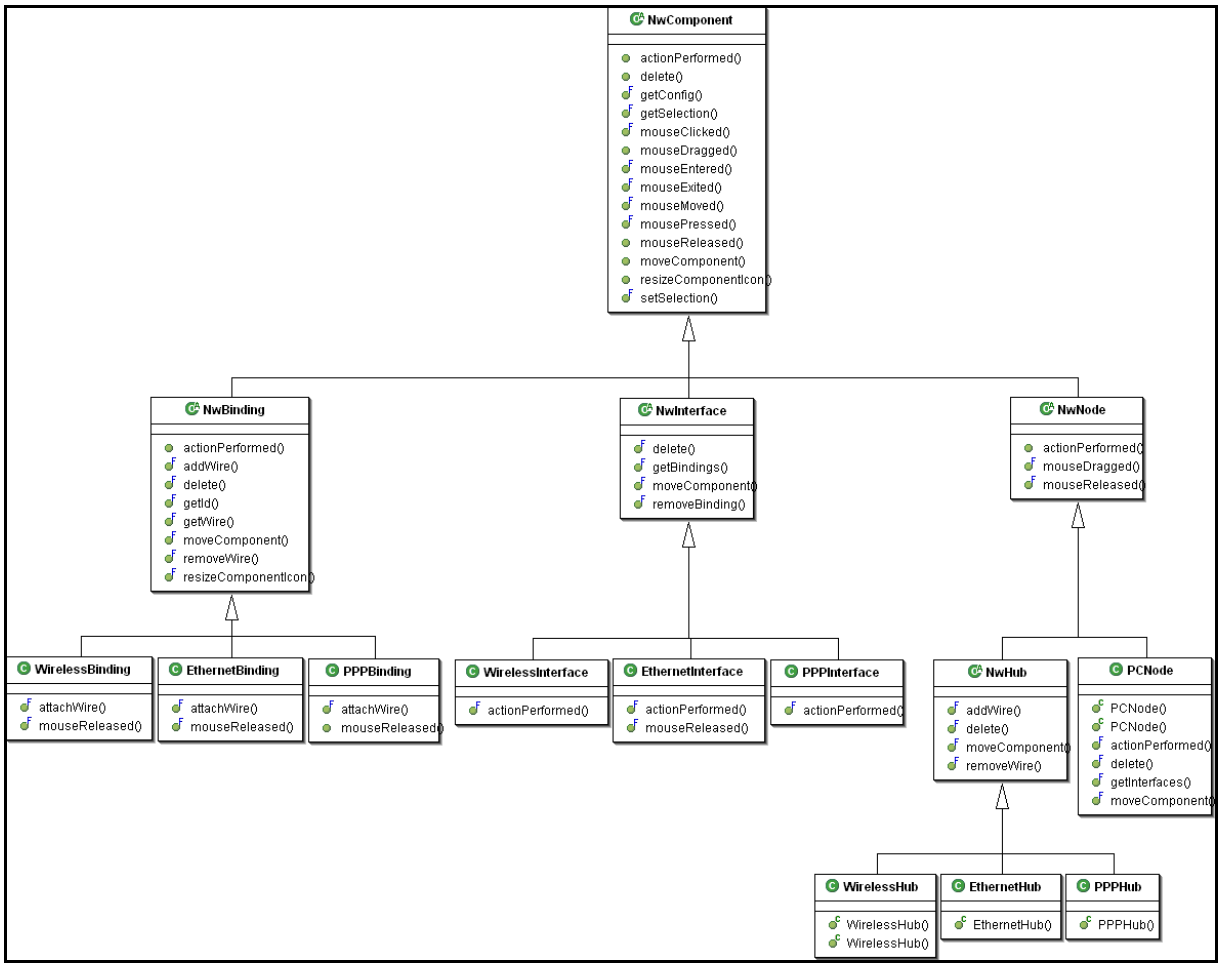


Abbildung 16: Klassenhierarchie von Netzwerkkomponenten

Auf der zweiten Hierarchiestufen finden sich drei Klassen (*NwBinding, NwInterface, NwNode*), welche die Oberklassen für Bindings, Interfaces resp. Knoten repräsentieren. Diese drei Klassen sind zwar auf der selben Hierarchiestufe, jedoch besteht zwischen ihnen eine logische Hierarchie (A*bb. 17*). Dies rührt daher, dass ein Knoten mehrere Interfaces besitzen kann, ein Interface über mehrere Bindings verfügen kann, ein Binding genau eine oder keine Verbindungslinie aufweist und diese gegebenenfalls in einem Hub mündet. Jede Klasse in dieser Kette muss dementsprechend seine Nachbahren kennen. Zum Beispiel besitzt die Klasse *NwBinding* ein Feld *owner*, welches die dazugehörige Schnittstelle speichert und ein Feld *myWire*, welches die Verbindungslinie hinterlegt. Zudem bietet jede dieser Klassen (mit Ausnahme von *NwNode*, wo die Unterklassen dies erledigen) Methoden zum Verschieben, Löschen und Entfernen entsprechender Komponenten an.

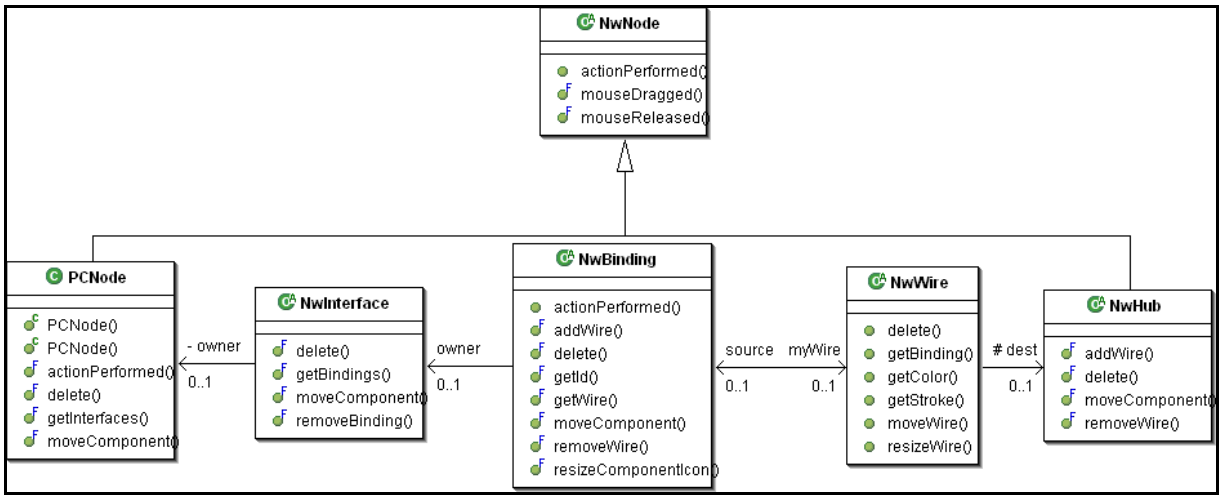


Abbildung 17: logische Hierarchie von Netzwerkkomponenten

Die beiden Klassen *PCNode* und *NwInterface* haben zudem die Aufgabe, Positionen für künftige Schnittstellen resp. Bindings zu berechnen. Dazu verwenden Beide einen einfachen Algorithmus, welcher im Quellcode näher beschrieben ist.
Auf der dritten resp. vierten Hierarchiestufe befinden sich die konkreten Implementationen der Netzwerktypen. Jeder Netzwerktypus muss eine Binding, eine Interface, eine Hub sowie eine Klasse für Verbindungslinien besitzen, welche die entsprechenden Oberklassen erweitern. Diese Klassen legen meist nur Eigenschaften wie Farbe, Muster und Icon fest.
Eine weitere wichtige Klasse im Configurationpackage ist *NwWire* (*Abb. 18*). Sie ist die Oberklasse sämtlicher Verbindungslinien und implementiert wichtige Methoden zur Manipulation jener Verbindungen. Bei den Methoden handelt es sich beispielsweise um eine Vergrösserung, eine Verschiebung oder eine Löschung der Verbindungslinie. Auffallend ist, dass die Unterklassen keine Methoden aufweisen. Sämtliche Unterklassen verfügen nur über einen Konstruktor, in welchem die Farbe, die Liniendicke sowie das Linienmuster festgelegt wird.

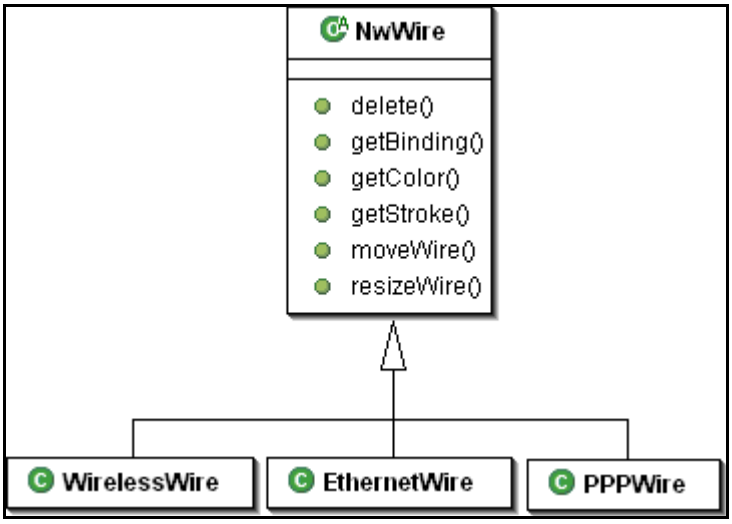


Abbildung 18: Hierarchie der Verbindungslinien

2.2.3 Erweiterungsmöglichkeiten

Im Definitionsmodus liessen sich diverse Erweiterungen (funktionale, sowie benutzerunterstützende) realisieren. In der folgenden Liste sind ein paar Möglichkeiten aufgeführt:

- Die Konfiguration der Komponenten könnte miteinbezogen werden. D.h. es sollten Mechanismen zur Verfügung gestellt werden, damit das Netzwerk konfiguriert werden kann. Beispielsweise könnte mit einem Doppelklick auf einer Komponente ein Menu erscheinen, welches es erlaubt Eigenschaften der Komponenten wie z.B. Netzwerkadressen, Namen etc. festzulegen.
- Weitere Netzwerktypen könnten hinzugeführt werden. Beispielsweise könnten die Netzwerke *Token Ring*, digitale Telephonnetze (*ISDN*), *Bluetooth* etc. implementiert werden.
- Angenehm wäre, wenn der Benutzer seine Aktionen jeweils rückgängig machen könnte. Dies könnte mithilfe eines Design-Pattern erreicht werden.
- Die Standardeinstellungen, wie z.B. die Icons, die Linienfarbe, die Liniendicke etc. könnten über einen Menueintrag (*Preferences*) verändert werden.
- Die Knotengrösse sollte beliebig sein und dem Benutzer sollte es möglich sein, diese selbst zu verändern. Zudem wäre wünschenswert, wenn die Interfaces und die Bindings beliebig verschoben werden könnten.
- Die Verbindungslinien sollten derart erweitert werden, sodass sie gekrümmt und gebogen werden können.
- Ein Hilfsassistent sowie ein Tutorial könnten hilfreiche Erweiterungen darstellen.
- Die Konfiguration sollte ausgedruckt werden können.
- Schwierig aber schön wäre ein Layoutmanager, der die Komponenten versucht automatisch anzuordnen. Dies wäre dann hilfreich, wenn Konfigurationen importiert werden, welche nicht im GUI erstellt wurden und dadurch auch keine Koordinaten in ihrer Definition aufweisen.
- Eine durchaus brauchbare Erweiterung wäre eine Bibliothek mit bereits vorkonfigurierten Komponenten. Es sollte auch möglich sein, einzelne konfigurierte Komponenten dieser Bibliothek zuzufügen.
- An vielen Stellen sollten die Fehler noch besser abgefangen und entsprechend reagiert werden.

2.2.4 Bekannte Probleme

Meiner Meinung nach, ist die Implementierung der Zoom-Funktion nicht optimal. Beispielsweise ist es nicht möglich, eine bestehende Netzwerkdefinition zu laden, wenn in die Vergrösserung nicht 100% beträgt (Dies ist der Grund, weshalb beim Laden der Zoomfaktor auf 100% zurückgesetzt wird).
Ein weiteres Problem tritt mit dem Seitenfenster auf. Werden verschiedene Knoten gleichzeitig angewählt, wird nur einer im Seitenfenster aufgezeigt. Ebenfalls wird beim Löschen eines Knotens die Definitionsdatei nicht aus dem Seitenfenster entfernt.

2.3 Der Simulationsmodus

Der Simulationsmodus verfolgt das Ziel Testergebnisse visuell darzustellen. Diese Visualisierung hilft dem Benutzer Sicherheitslücken und Funktionsmängel besser zu erkennen. Ohne den Simulationsmodus müsste der Anwender eine lange menschen-unverständliche Datei durchlesen, wobei mit grosser Wahrscheinlichkeit nicht alle Mängel entdeckt würden.

2.3.1 Anwendung

Sobald das gewünschte Netzwerk im Definitionsmodus erstellt wurde, kann anhand des Knopfes *Simulation* in den Simulationsmodus gewechselt werden. Nun werden die Menueinträge verändert und die Toolbar ersetzt.

2.3.1.1 Die Menuleiste

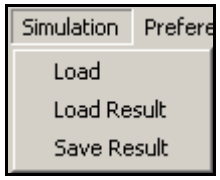


Abbildung 19: Menu

Innerhalb der Menuleiste finden sich drei Einträge. Die beiden Einträge *Preferences* und *About* sind unverändert und reagieren dementsprechend analog zum Definitionsmodus. Der Eintrag *Simulation* hingegen ist neu dazugekommen. In seinem Untermenu sind drei Einträge vorhanden (*Abb. 19*). Mit *Load* können Testdateien geladen werden, *Load Result* lädt Ergebnisse von bereits durchgeführten Simulationen und mit *Save Result* können Testergebnisse abgespeichert werden.
Wird *Load* gedrückt, erscheint ein Datei-Browser und der User kann eine Testdefinitionsdatei in die Applikation reinladen. Anschliessend erscheint eine Dialogbox, wo der Benutzer eine Endzeit eingeben muss. Nun wird der *Simulator* aufgerufen und ihm wird die Testdefinitionsdatei und die Endzeit überreicht. Wird *Load Result* gedrückt, erscheint wiederum ein Datei-Browser, aber diesmal muss eine Ergebnisdatei eines bereits durchgeführten Test reingeladen werden. Mit *Save Results* kann der durchgeführte Test abgespeichert werden. Dazu öffnet sich erneut ein Datei-Browser und der User kann nun den gewünschten Dateinamen eingeben (die Endung .xml muss ebenfalls angefügt werden).

2.3.1.2 Die Werkzeugleisten

Die Werkzeugliste besteht aus vier Komponenten (*Abb. 20*). Die erste Komponente dient zur Steuerung der Simulation. Sie besitzt die fünf Knöpfe *Play, Pause, Stop, Step forward, Step backward*, wobei mit *Play* der Test gestartet wird, mit *Pause* die Simulation unterbrochen wird, mit *Stop* die Simulation ganz zurücksetzt wird und *Step forward* resp. *Step backward* eine Zeiteinheit nach vorne resp. hinten springt und anschliessend eine Zeiteinheit abspielt.

Abbildung 20: Werkzeugliste der Simulation

Die zweite Komponente regelt die Simulationsgeschwindigkeit. Der Regler kann mit der Maus verschoben werden. Dabei gilt: Je weiter links der Regler ist, desto langsamer wird die Simulation.

Die Dritte Komponenten zeigt den Fortschritt des Ablauf an. Der Regler kann ebenfalls verschoben werden, sodass ein Sprung in der Simulation realisiert werden kann. Das Feld neben dem Fortschrittsbalken, zeigt die jeweilige logische Zeit an.
Die letzte Komponente ist der *Configuration* Knopf, welcher es erlaubt, zurück in den Definitionsmodus zu wechseln. Falls eine Simulation noch im Gange ist, wird diese abgebrochen.

2.3.1.3 Das Seitenfenster

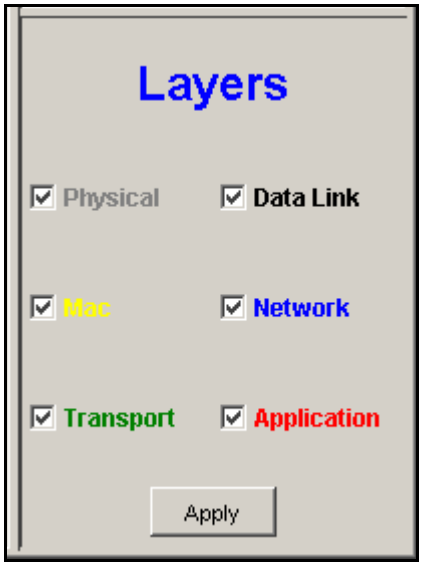


Abbildung 21: Ebenenfilter

Sobald eine Testdatei mit Hilfe des Menueintrages *Load* oder *Load Result* geladen wird, erscheint im unteren Teil des Seitenfensters ein weiteres Panel (*Abb. 21*). Dieses Panel erlaubt es, die Ereignisse der jeweiligen Ebenen zu filtern. Jedes Ereignis der Simulation ist einer der sechs Ebenen (*Physical, Data Link, Mac, Network, Transport, Application*) zugeordnet. Standardmässig werden sämtliche Ereignisse der Simulation angezeigt. Interessiert sich jedoch ein Benutzer lediglich für die Ereignisse auf spezifischen Ebenen, können diese Ereignisse herausgefiltert werden. Dazu müssen die Checkboxes der gewünschten Ebenen angekreuzt und diejenigen der anderen Ebenen abgekreuzt werden. Damit die Änderung auch stattfindet, muss anschliessend der *Apply* Knopf gedrückt werden. Nun werden nur noch jene Ereignisse der gewünschten Ebenen auf der Erstellfläche angezeigt. Anmerkung: Derzeit unterstützt der Simulator keine Ereignisse der *Physical*-Ebene.

2.3.1.4 Die Erstellfläche

Auf der Erstellfläche werden die Testresultate dargestellt. Ein Testresultat ist nichts anderes als eine Menge von Ereignissen. Ein Ereignis ist ein *XML*-Element mit dem Wurzelnamen *events*. Zudem besitzt es Attribute und weitere Unterelemente (*Abb. 22*).

```
<event time="1" node="diufpc55" layer="5" service="application" src="diufpc55" dst="diuflx01" id="unique1">
    <application program="wget" parameters="http://diuflx02/test.html" type="launch"/>
</event>
```

Abbildung 22: Beispiel von einem XML-Element eines Ereignisses

Jedes Ereignis gehört zu einer Ebene (Attribut *layer*), zudem besitzt es eine gewisse Simulationszeit (Attribut *time*) und einen Simulationsort (Attribut *node*). Bei der Simulationszeit handelt es sich um eine ganze, positive Zahl. Ein Ereignis wird genau dann dargestellt, sobald die Simulationszeit des Ereignisses mit jener Zeit des Simulationsfortschrittes übereinstimmt. Der Simulationsort ist abhängig von der Ebene des Ereignisses. Je nachdem welcher Ebene das Ereignis angehört, ist der Simulationsort ein Knoten, eine Schnittstelle oder eine Verbindungslinie.

2.3.1.4.1 Knoten

Im Simulationsmodus werden die Knoten als Simulationsorte verwendet. Sämtliche Ereignisse der Ebenen *Transport* und *Application* werden auf Knoten dargestellt. Sobald die Simulationszeit des Ereignisses mit jener des Simulationsfortschrittes korrespondiert, wird das entsprechende Ereignis anhand einer Animation angezeigt. Diese Animation ist in drei Phasen aufgebaut. Zuerst wird das Ereignis eine gewisse Zeit lang kontinuierlich vergrössert, anschliessend verbleibt es einen Moment lang in der erreichten Grösse, als Letztes wird es wieder verkleinert, bis es schlussendlich ganz verschwindet (*Abb. 23*). Treten mehrere Ereignisse gleichzeitig auf dem selben Knoten auf, werden diese in abwechselnder Reihenfolge gezeigt.

Abbildung 23: Animation auf Knoten

Ereignisse auf Knoten besitzen entweder Rot oder Grün als Hintergrundfarbe. Rot bedeutet, dass das Ereignis der *Application*-Ebene angehört, während Grün die Ereignisse der *Transport*-Ebene markiert. Der schwarze Schriftzug des Ereignisses wird aus dem dazugehörigen XML-element gewonnen. In unserem Beispiel besteht der Schriftzug aus den Attributen des A*pplication* Unterelementes (vgl. *Abb. 22* und *Abb. 23*).

2.3.1.4.2 Schnittstellen und Bindings

Schnittstellen, oder besser gesagt deren Bindings, dienen als Simulationsorte der Ereignisse der Ebenen *Mac* und *Network*. Ereignisse werden auch hier mit Hilfe einer Animation dargestellt. Die Animation verläuft ebenfalls in drei Phasen. Zuerst wird das Ereignis vergrössert und nach „aussen“ verschoben. Anschliessend verleibt es einen Moment lang in dieser Position, bis es schlussendlich wieder verkleinert und zurückverschoben wird (*Abb. 24*). Beim gleichzeitigen Auftreten verschiedener Ereignisse auf demselben Binding, werden diese ebenfalls alternierend angezeigt.

Abbildung 24: Animation auf Bindings

Ereignisse der Ebenen *Mac* und *Network* können ebenfalls optisch unterschieden werden. Die Ereignisse der *Mac*-Ebene, weisen ein Icon auf, welches das Wort *Mac* enthalten. Icons der *Network*-Ebene enthalten meist den Namen eines Protokolls, wie z.B. IP.

2.3.1.4.3 Verbindungslinien und Hubs

Auf den Verbindungslinien werden Ereignisse der *Data-Link*-Ebene Ebene dargestellt. Die Darstellung erfolgt auch hier mit einer Animation. Die Animation lässt ein schwarzes Quadrat auf der Verbindungslinie vom Sender zum allen Empfängern wandern (*Abb. 25*).

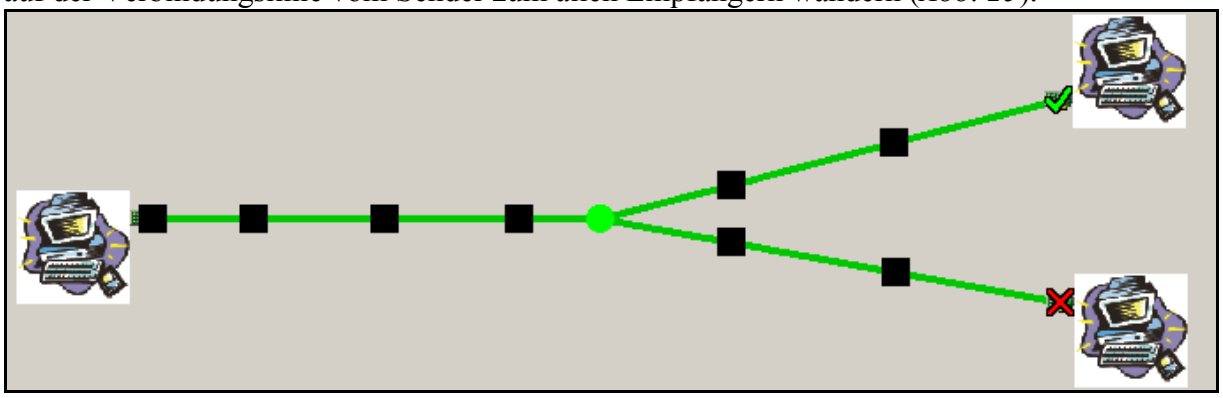


Abbildung 25: Animation auf Verbindungslinien

In diesem Beispiel ist der Sender der Knoten links und die beiden Empfänger sind auf der rechten Seite angesiedelt. Das schwarze Quadrat stellt ein Datenpaket dar und wandert von Links nach Rechts. Sobald es beim Hub angelangt ist, wird es multipliziert und an die entsprechenden Empfänger weitergeleitet. Diese Multiplikation wird in Wahrheit nicht von Hub ausgelöst, sondern vom *Scheduler*. Der *Scheduler* entfernt das ankommende Ereignis, da dieses nun seine Endzeit erreicht hat. Unmittelbar darauf, speist der *Scheduler* jedoch die neuen Ereignisse in die Simulation ein, zu welchen auch die Empfangsereignisse gehören. Wichtig ist, dass es sich optisch gesehen nur um ein einiges Datenpaket handelt, während sich in der Simulationsdatei mehrere Einträge finden.
Erreicht ein Datenpaket den Empfänger, bleibt es auf dem Binding stehen und verwandelt sich entweder in ein Gutzeichen oder in ein Kreuz. Das Gutzeichen bedeutet, dass das Datenpaket für diesen Empfänger gedacht war, während mit dem Kreuz angedeutet wird, dass das Datenpaket zwar empfangen wurde, jedoch nicht für diesen Knoten gedacht war und auf *MAC*-Ebene verworfen wird.

2.3.2 Architektur

In diesem Abschnitt wird beschrieben wie der Simulationsmodus aufgebaut ist. Dazu wird zuerst eine Übersicht über das Package sowie seine Klassen gegeben. Anschliessend folgt ein Kapitel über den Aufbau des Moduls.

2.3.2.1 Das Package und seine Klassen

Das zum Simulationsmodus dazugehörige Package heisst *Simulation* und befindet sich im Ordner *src/Simulation*. Es umfasst 12 Klassen (*Abb. 26*), welche sämtliche Funktionalitäten der Simulation realisieren. Ein Grossteil der Klassen repräsentieren Ereignisse. Für jede Ereignisebene existiert eine separate Klasse, welche allesamt die abstrakte Klasse *SimuEvent* erweitern.
Die drei Klassen *SimulationScheduler, SimulationToolbar und SimulatorThread* haben eine wichtige Rolle, denn sie koordinieren den Simulationsablauf.
Eine weitere wichtige Klasse ist *LayerFilterPanel*, welche das Panel in unteren Teil des Seitenfensters repräsentiert (*siehe Abschnitt Seitenfenster des Simulationsmodus*).
SimulationException ist eine Klasse, welche von *java.lang.Exception* erbt. Sie ist demnach eine Fehlerklasse und wird nur dann erstellt, wenn Netzwerkkomponenten nicht gefunden werden können.

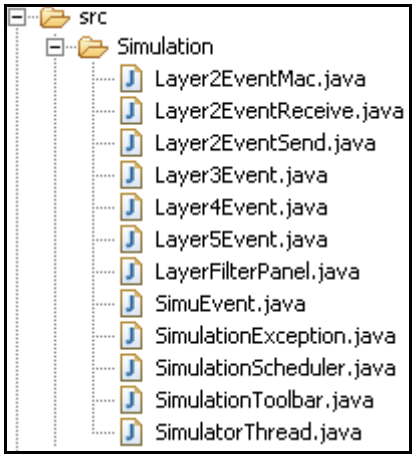


Abbildung 26: Klassen

2.3.2.2 Klassen: Hierarchie und Zusammenspiel

Im Simulatormodus existiert genau eine wichtige Hierarchie, die nun beschrieben werden soll (*Abb. 27*).

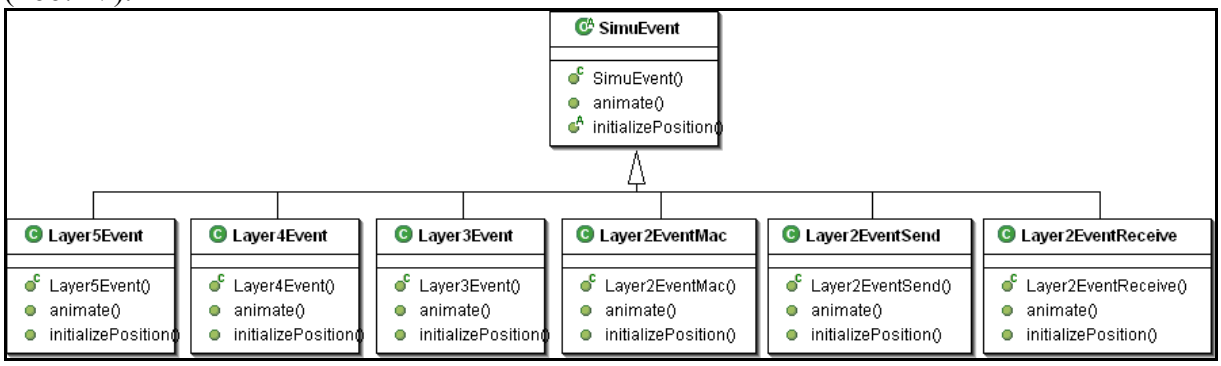


Abbildung 27: Ereignishierarchie

An der Spitze der Hierarchie findet sich die Klasse *SimuEvent*. Diese Klasse ist abstrakt und besitzt zwei Methoden und eine geringe Anzahl an Felder. Die eine Methode *animate*, regelt die Sichtbarkeit der Ereignisse, wenn diese gleichzeitig und an selber Stelle auftreten. Die andere Methode *initializePosition* ist abstrakt und verfolgt die Absicht, die Anfangsbedingungen der Ereignisse zu setzen. Bei diesen Anfangsbedingungen handelt es sich um die Grösse und Position der jeweiligen Ereignissen. Das wichtigste Feld dieser Klasse ist das XML-Element *event*, wodurch das Ereignis vollständig charakterisiert wird. Das graphische Erscheinungsbild wird praktisch nur aus diesem Feld hergeleitet (Eine Ausnahme

bilden die Ereignisse der *Data-Link*-Ebene, da diese auch eine Endzeit besitzen, welche aus dem dazugehörigen Empfangs-Ereignis errechnet werden muss).
Die beiden Klassen *Layer5Event* und *Layer4Event* sich praktisch identisch. Beide besitzen einen Konstruktor, welcher die Anfangsposition errechnet. Zudem wird der Text des Ereignisses aus seinem *event* Feld gewonnen. Die Methode *animate* wird derart erweitert sodass die weiter oben beschriebene Animation stattfinden kann (vgl. Abschnitt *Knoten*). Der einzige Unterschied zwischen den beiden Klassen ist die Farbe und der Text der Ereignisse.
Die Klassen *Layer3Event* und *Layer2EventMac* sind ebenfalls praktisch identisch. Auch sie errechnen im Konstruktor die Anfangsbedingungen und leiten das entsprechende Icon aus dem *event* Feld ab. Die *animate* Methode wird ebenfalls erweitert, sodass die oben beschriebene Animation stattfinden kann (vgl. Abschnitt *Schnittstellen und Bindings*).
Die Klassen *Layer2EventSend* und *Layer2EventReceive* sind ebenfalls sehr ähnlich. Im Konstruktor wird die Anfangsposition und die Sendegeschwindigkeit errechnet. Die Animation der Klasse *Layer2EventReceive* ist ein wenig komplizierter als jene von *Layer2EventSend*. In dieser Animation müssen nämlich noch die Bilder bei der Ankunft beim Empfänger eingefügt werden.
Damit die Simulation abläuft, sind weitere Klassen notwenig. Die drei Klassen, welche für die Koordination des Testablaufs verantwortlich sind, heissen *SimulationToolbar, SimulatorThread und SimulationScheduler.* Da diese Drei zusammen das Herz der Simulation bilden, soll ihr Zusammenspiel im folgenden Abschnitt erläutert werden.
Die Klasse *SimulaitonToolbar* stellt das Kontrollzentrum der Simulation dar. Sie beinhaltet die fünf Knöpfe (*play, pause, stop, step forward, step backward*) und die beiden Regler *Speed* resp. *Progress*. Speziell wichtig ist der Knopf *play* und der Regler *Progress*. Mit *Play* wird der *SimulatorThread* angeworfen und mit *Progress* werden jeweils die aktuellen Ereignisse in den *SimulationScheduler* geladen. Alle anderen Knöpfe basieren entweder auf diesen Beiden, oder verändern lediglich ein Feld (z.B. das Pause oder Stopfeld) der Klasse *SimulatorThread*. Der Simulationsfortschritt geht also zwangsläufig über den *Progress* Regler, was die Komplexität reduziert.
Die *SimulatorThread* Klasse ist abgeleitet von *java.lang.Thread*. Mit anderen Worten: Sie ist ein separater Thread der Applikation. Von grosser Wichtigkeit sind die beiden Methoden *run* sowie *progress*. *run* ruft in regelmässigen Abständen die Methode *animateEvents* der Klasse *SimulationScheduler* auf, wodurch die Ereignisse animiert werden. Zudem ist *run* verantwortlich, dass die Simulation auf allen Computern gleich schnell abläuft. Gelegentlich ändert sie auch den Fortschritt der Simulation, indem sie den *Progress* Regler der *SimulationToolbar* versetzt. Diese Änderung des *Progress* Reglers bewirkt, dass die Methode *progress* der Klasse *SimulatorThread* aufgerufen wird. Die Methode *progress* ruft die Methode *LoadEvents* der Klasse *SimulationScheduler* auf, welche die derzeitigen Ereignisse vom *SimulatorScheduler* entfernt und die nun Aktuellen nachlädt (vgl. Abb. 28).

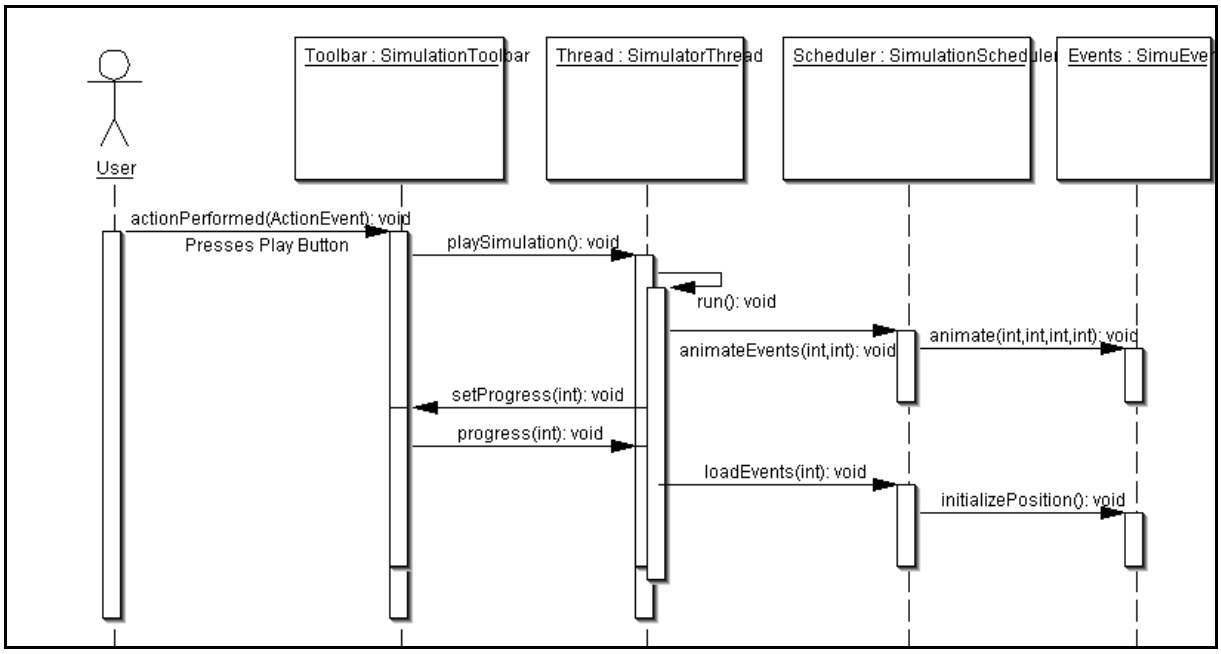


Abbildung 28: Das Kernkonzept des Simulators

Abschliessend soll noch die Arbeitsweise vom *SimulationScheduler* erläutert werden. Der *SimulationScheduler* legt zwei Tabellen an. Die Eine (*eventTable*) enthält sämtliche Ereignisse, welche in der Simulation auftreten. Die andere Tabelle (*activeEvents*) enthält lediglich die Ereignisse, welche derzeit aktiv sind. Beide Tabellen enthalten die Spalten *Starttime, Endtime, Location* und *Simulatorevent*. *Starttime* gibt Auskunft wann das Ereignis dargestellt werden soll, *Endtime* enthält der Zeitpunkt des Endes der Darstellung, *Location* besagt, wo das Ereignis erscheinen soll und *Simulatorevent* enthält eine Referenz auf die entsprechende Ereignisklasse. *activeEvents* enthält zusätzlich noch die Spalten *myPhase* und *TotalPhase*. *TotalPhase* gibt an, wie viele Ereignisse derzeit gleichzeitig auf dem selben Ort dargestellt werden müssen. *myPhase* enthält die Phasennummer, welcher das Ereignis angehört. Diese beiden Spalten werden benötigt, damit ein Rotationsplan der betroffenen Ereignissen in der Klasse *SimuEvent* aufgesetzt werden kann.

2.3.3 Erweiterungsmöglichkeiten

Im Simulationsmodus könnten diverse Erweiterungen realisiert werden. Die folgende Liste enthält sowohl funktionale wie auf Benutzerunterstützende Erweiterungen.

- Die Definition von Test sollte direkt im Simulationsmodus möglich sein. Beispielsweise könnte der Benutzer zwei Knoten auswählen und anschliessend könnte er eine Liste von Tests zwischen jenen Knoten erstellen.
- Auf den Ereignissen könnte ein Tooltip erscheinen, welcher das XML-Element anzeigt.
- Für die Ereignisse der Ebenen *Transport* und *Application* könnten Icons anstelle von Text verwendet werden.
- Die Datenpakete auf den Verbindungslinien könnten verschiedene Farben besitzen, welche durch die Paketidentität berechnet würden. So wäre es möglich, Datenpakete optisch unterschiedlich zu gestalten.
- Ereignisse der *Physical* Ebene könnten auf den Hubs dargestellt werden.

2.3.4 Bekannte Probleme

Mir sind nur zwei Probleme bekannt. Das Eine tritt auf, wenn ein Ereignis der *Data-Link-*Ebene empfangen wird, wobei das schwarze Quadrat mit einem Gutzeichen resp. einem Kreuz ersetzt wird. Wird zur gleichen Zeit auf den *step backward* Knopf gedrückt, springt das Datenpaket zwar zurück, jedoch wird das Gutzeichen resp. Kreuz nicht wieder durch ein schwarzes Quadrat ersetzt (*Abb. 29*).

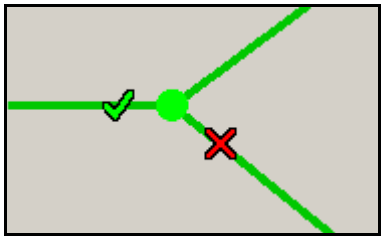


Abbildung 29: Iconfehler

Das zweite Problem ist sehr unregelmässig, aber hat keinen direkten Einfluss auf den Ablauf der Simulation. Gelegentlich erscheint der Fehler aus (*Abb. 30*) in der Konsole. Dieser Fehler kommt meines Erachtens von einer Paint Methode. Es könnte gut sein, dass er mit einer späteren Java Version verschwinden wird.

```
java.lang.Exception: Stack trace
        at java.lang.Thread.dumpStack(Thread.java:1064)
        at javax.swing.SwingGraphics.createSwingGraphics(SwingGraphics.java:147)
        at javax.swing.JComponent._paintImmediately(JComponent.java:4670)
        at javax.swing.JComponent.paintImmediately(JComponent.java:4488)
        at javax.swing.RepaintManager.paintDirtyRegions(RepaintManager.java:410)
        at javax.swing.SystemEventQueueUtilities$ComponentWorkRequest.run(SystemEventQueueUtilities.java:117)
        at java.awt.event.InvocationEvent.dispatch(InvocationEvent.java:178)
        at java.awt.EventQueue.dispatchEvent(EventQueue.java:454)
        at java.awt.EventDispatchThread.pumpOneEventForHierarchy(EventDispatchThread.java:201)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:151)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:145)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:137)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:100)
java.lang.NullPointerException
        at javax.swing.JComponent._paintImmediately(JComponent.java:4671)
        at javax.swing.JComponent.paintImmediately(JComponent.java:4488)
        at javax.swing.RepaintManager.paintDirtyRegions(RepaintManager.java:410)
        at javax.swing.SystemEventQueueUtilities$ComponentWorkRequest.run(SystemEventQueueUtilities.java:117)
        at java.awt.event.InvocationEvent.dispatch(InvocationEvent.java:178)
        at java.awt.EventQueue.dispatchEvent(EventQueue.java:454)
        at java.awt.EventDispatchThread.pumpOneEventForHierarchy(EventDispatchThread.java:201)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:151)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:145)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:137)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:100)
In paintImmediately null graphics
```

Abbildung 30: Paintfehler?

2.4 Aufbau und Architektur der graphischen Oberfläche

In den beiden vorherigen Abschnitten wurden die Packages *Configuration* und *Simulation* diskutiert. Nebst diesen beiden Packages braucht es noch weitere Klassen um die Packages zusammenzufügen. Im Ordner *src* finden sich deshalb noch vier Klassen. Die drei Wichtigsten sollen hier kurz beschrieben werden.
Die drei Klassen *Analyser, DrawPanel und InfoSplitPane* sind Behälter. Sie repräsentieren jeweils eine graphische Komponente (*Abb. 31*).

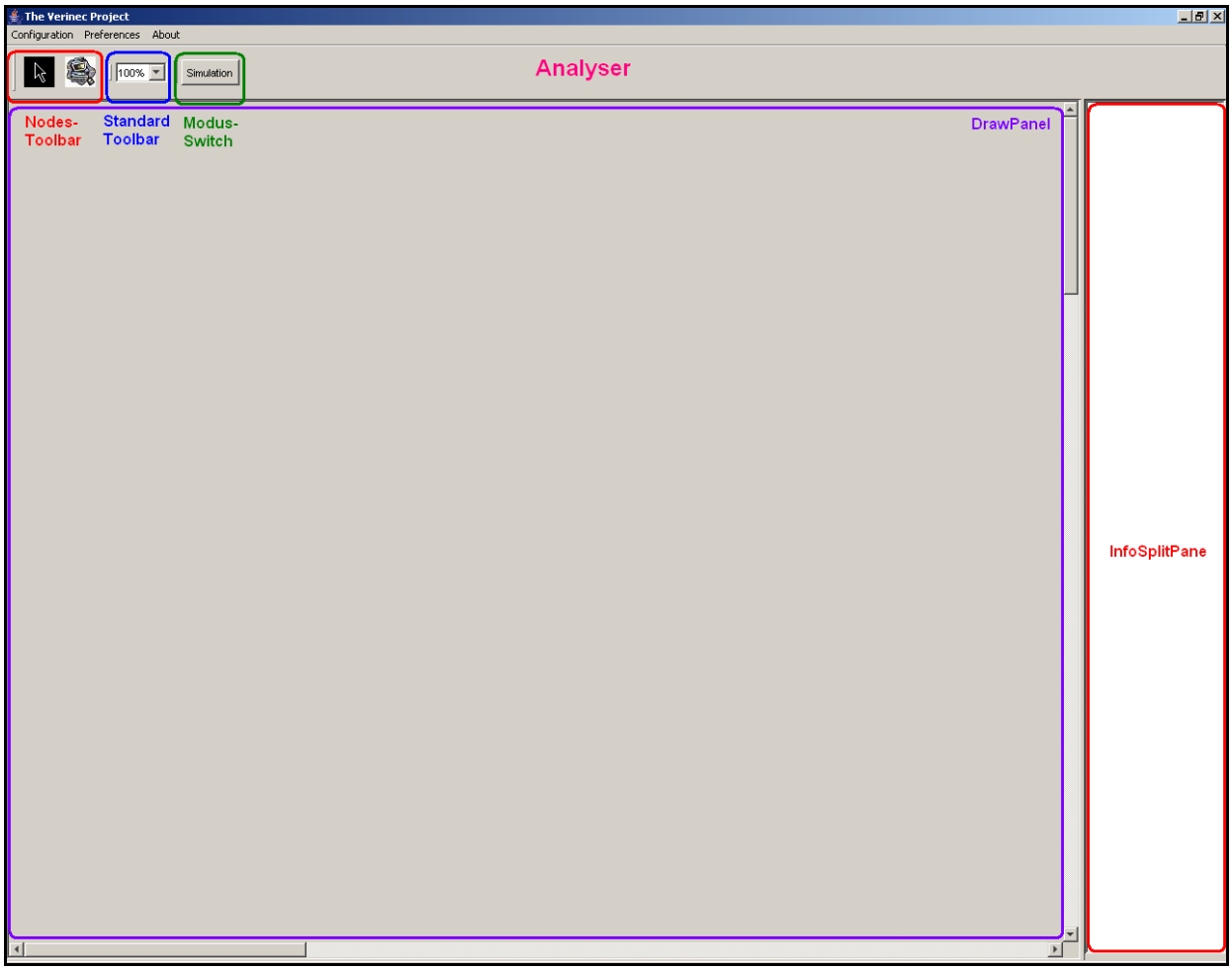


Abbildung 31: die Behälterklassen

Die *Analyser* Klasse verfügt über die *Main*-Funktion. Zudem verkörpert sie das Hauptfenster der Applikation. Darüber hinaus verwaltet sie den aktuellen Modus und verarbeitet die Ereignisse der Menuleiste.
Die *DrawPanel* Klasse repräsentiert die Erstellfläche. Zu der Aufgaben der Erstellfläche zählt das Markieren der Komponenten, das Zeichnen der Verbindungslinien, sowie das Weiterleiten von Informationen an andere Komponenten, welche keine direkte Referenz zueinander besitzen.
Die letzte der drei Klassen heisst *InfoSplitPane* und stellt das Seitenfenster dar. Die Aufgaben des Seitenfenster beschränken sich auf die Ausgabe von Knoteninformationen sowie das Filtern der Simulationsereignisse im Simulationsmodus.

3 Bilanz und Kritik

Insgesamt bin ich mit dem Projekt zufrieden. Einen besonderen Gefallen fand ich am Software Engineering, am freien Arbeiten, sowie an der guten Arbeitsumgebung. Einen Dank geht hier an meine beiden Assistenten Dominik Jungo und David Buchmann, die bei Fragen sowie bei Vorschlägen immer ein offenes Ohr hatten.
Verbesserungsfähig wäre der Rahmen des Projekts, die Definition der Aufgaben, sowie die zeitliche Organisation. Leider veränderten sich die Gegebenheiten des Projektes während dessen Realisierung, sodass vielerorts Stellen nachgebessert oder gar aufgegeben wurden.

4 Referenzen

- Andrew S. Tanenbaum. Computer Networks. Prentice Hall. 4th Edition. 2003
- Jdom. www.jdom.org/
- Junit. www.junit.org
- VeriNeC. diuf.unifr.ch/tns/verinec
- Desmoj. Simulationsframework. http://www.desmoj.org/
- Dominik Jungo, David Buchmann, Ulrich Ultes-Nitsche. The Role of Simulation in Network Configuration Engineering Approach. 2004
- David Buchmann. VeriNeC Translation Module. Working Paper, 2004
- Dominik Jungo. Theoretical Aspects of discrete Event-Based Simulation. Working Paper, 2004
- Diverse Bilder: http://www.smok.se/images/paket.gif , http://www.t-english.com/courses/images/paket.gif, http://scna.com/smartpeople/internet/images/paket.jpg