Implementation of a Web-based Interface for Virtual Router Configuration

Diplomarbeit
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von:
Eveline Kurt

2002

Leiter der Arbeit:
Prof. Dr. Torsten Braun

Forschungsgruppe Rechnernetze und Verteilte Systeme (RVS)
Institut für Informatik und angewandte Mathematik

Contents

Chapter 1

Introduction

This thesis presents the implementation of a web-based interface for configuring virtual routers.

As an introduction we will describe an application area for this work. The application can be used for remote teaching, an application field which is becoming more and more important. More specifically we will present the possibilities to use it in a teaching module based on IP network simulation.

In the next chapter we will present the underlying virtual router application and describe how its properties can be used for a IP Network Simulation module. Next, we describe the application itself, from the point of view of a user. The following chapters will then focus on the design and the implementation of the application. In the last chapter we present the strengths and weaknesses of the application and the possibilities to extend it.

Chapter 2

A Virtual Internet and Telecommunications Laboratory

2.1 Introduction

In the first world, the Internet is widely used for business applications, communication and entertainment. As a natural consequence we also want to use it for teaching and thus learning purposes. Many institutions and businesses already offer web-based teaching. Especially schools teaching computing sciences should take the opportunity to use the media they help to develop. Several informatics institutes of Swiss universities participate in the Swiss Virtual Campus programs [SVC]. One of the projects of the SVC is VITELS(Virtual Internet and Telecommunications Laboratory of Switzerland) [VIT]. Seven different distant learning modules are being developed in the VITELS project. A module consists of a distant learning unit, allowing a student to access remote resources and use them to perform exercises. The unit may be accessed remotely from any web browser. The modules are the following: Linux System Installation and Configuration, IP Network Simulation, Configuration and Evaluation of a real IP Network, Client\Server programming, Protocol Analysis, IP Security and Firewall Management.

This thesis presents an application that can be used for the Network Simulation module.

2.2 IP Network Simulation Module

The teaching aims of the IP Network Simulation Module for a student could be the following:

A user should learn to plan the configuration of a network on the IP Network

level. This includes the setting of routing tables, the configuration of interfaces, including the setting of IP addresses, the use of subnet- and bitcount masks. To get most out of this exercise a user should also learn to differentiate between classful and classless subnetting and different network classes.

The module should also simulate very basic facts, like routers have to be connected before they can forward packets and each router has a console as an interface over which it can be configured.

2.3 A Web-based Interface for Virtual Router Configuration (Wivrec)

For the IP Network Simulation Module we wanted to develop a web-based tool in order to enable student users to configure virtual routers (VR's) [1] remotely, display established VR connections and perform interconnectivity tests using TCP/IP applications or tools such as ping, traceroute, etc.

The application should have a generic interface to virtual routers and allows students to allocate the required resources to perform the virtual router exercise. Access to the resources should only be possible to authorized users. The graphical interface shall generate the number of requested VR's for the user and interconnect these VR's in order to build the desired topology. The configuration of such an topology can be stored for later use, so that the exercise can be interrupted. After such a VR topology has been generated and after the required computing and network resources have been allocated, the student can start to configure each VR. For this purpose a VR can be selected and a virtual configuration interface appears. This interface looks like a ordinary router terminal. Interface and routing table commands are supported. The configuration of a VR is saved, so that the exercise to be interrupted and continued later.

[1] we will use the following terms: virtual routers, VR's, microvar routers, router processes

Chapter 3

Virtual Routers

3.1 Overview

In this chapter we will present microvar, the application implementing the Virtual Routers. The microvar application underlies the wivrec application, which is the application described in this thesis. We will present many important microvar features, but this description is of course not exhaustive. For a thorough presentation you may refer to the publications which we used to plan this work and especially to write this chapter. [BBb]

3.2 Introduction

Microvar was designed to support and perform network simulations. It supports including real hosts in the simulations and thus to use real network traffic in the simulations performed with it.

An emulated topology consists of connected nodes. The nodes of an emulated topology can be real hosts or stand-alone processes (VR). The processes are composed of several components from the microvar architecture. A VR is connected to a real host over a softlink device, which is also a component of the microvar architecture. The softlink device conveys the network traffic from the host to the VR and vice versa. The VR is transparent to the host, it is viewed from the kernel and the user space as a (common) Ethernet device and it is also accessed like an Ethernet device.

The VR's can be connected by FIFO's [1] when they reside on the same host or they are connected using UDP tunnels when the two VR's to be connected are on different hosts. The main task of the VR's is to forward IP packets. For a topology example see fig. 3.1. It depicts two routers connected to an softlink

[1] A FIFO (also called named pipe) is a special file type that permits independent processes to communicate

device and connected to each other and two routers connected over and UDP tunnel on different hosts.

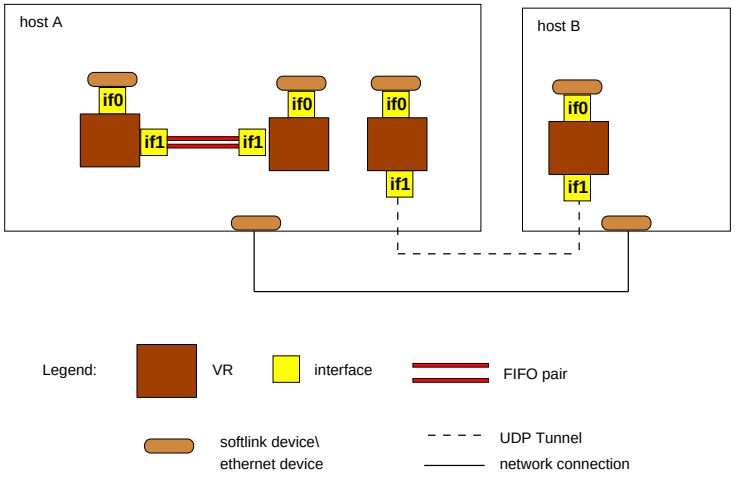

Figure 3.1: Example of a small topology

3.3 Architecture

The components the architecture of microvar is composed of are the IP Layer, the packet forwarding system, the Capsule Interpreter and the interfaces. The components of a VR are depicted in figure 3.2 [2]

Interface

An interface component is used between a router an a softlink device and between a router and connection elements like FIFO's. All the traffic a router forwards enters a router over an interface and leaves it through an interface. An interface itself is composed of may components.
The interface is composed of a queuing system, an IP translation unit (NAT), a Token Bucket Filter, and a UDP/softlink/FIFO connector.

IP Translation Unit

To route TCP traffic over a virtual from a source to a sink on the same (real)host, the IP translation unit is used. A TCP connection between virtual routers has to be emulated, TCP is not implemented in the protocol stack of the VR. For a more thorough explanation refer to [BBb].

[2] the figure is taken from [BBa]

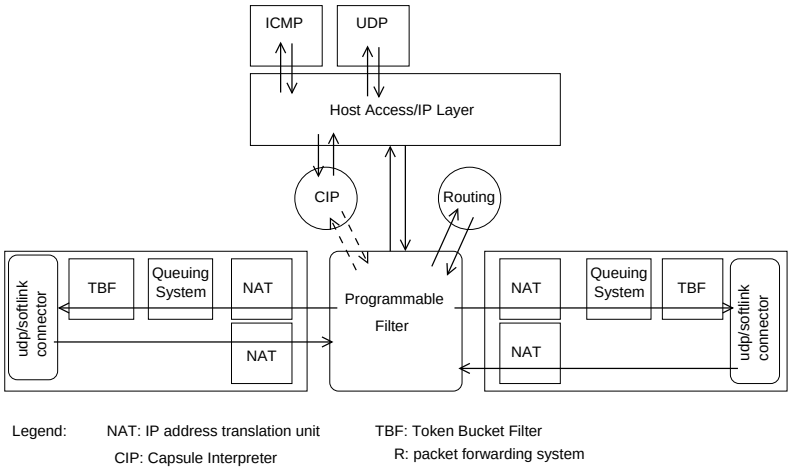

Figure 3.2: Architecture overview

Token Bucket Filter

The Token Bucket Filter limits the maximum bandwidth of an interface.

Programmable Filter, Host Access/IP Layer

The interfaces are attached to a Programmable Filter. This filter uses the packet forwarding system to forward the packets and is used and configured by CIP, the Capsule Interpreter. The Programmable Filter is relayed to the Host Access/IP Layer. This Layer has the protocol stacks UDP and ICMP as interface for the programs of the virtual host.

An arriving packet traverses NAT and reaches the Programmable Filter. There it might be processed by a Filter and forwarded to the IP Layer or routed to the appropriate interface. In this interface the packet traverses NAT again, traverses the queuing system, the Token Bucket Filter and continues its route. [3]

The Queuing System

The queuing system is composed of many components like queues, filters, shapers, schedulers. The application has the components Round Robin Early Detection Queue, a Weighted Fair Queuing, a RED queue with three drop precedences, and many more. The queuing system can be configured completely at runtime, so that the queuing mechanism can be modified while the virtual hosts are running. [BBb]

[3] there are also other possibilities, this is an example of one possibility

The Routing Mechanism

Packets are forwarded by the packet forwarding system which uses standard routing mechanisms but is extended to use source addresses, port numbers, protocol fields and ToS values.

Loadable Objects

Loadable Objects make the application very extensible. The Objects can be loaded at runtime, using the appropriate router configuration command. There are mainly two loadable objects implemented, those are the objects ping and traceroute.

3.4 Configuring a Microvar Router

The virtual host can be configured at runtime over a command line interface (CLI) or via an API [FB]. The command line interface is a shell which parses the commands and sends those commands to the API. This shell does not add any functionality to the API, it is just an interface for a human user. The command line interface can be accessed via console or telnet.

3.5 Microvar Extensions for IP Network Simulation

Microvar was designed to analyze IP traffic and can be used for testing DiffServ scenarios or even to emulate active networking. Microvar routes were not explicitly designed to model the setting of routing tables, interfaces. Setting the routing tables and interfaces is more a utility to set up a test network.

The requirements for IP Network Simulation Module are different. We want to model (also) the setting up of the hardware components of a network. The setting up of a network has to map the reality in time, e.g. a user has to wire the hosts before configuring them, and in space, e.g. a user enters a command to a router into the console representing this router.

An Example of Extending the Microvar Functionality to Ease its Use

We wanted it to be possible to send a ping command from every router. On the microvar level this required to connect a VR to a softlink device and to use the real hosts ping command to ping the softlink device. The softlink device and its IP number together with the IP number of the network containing the ping target had to be entered to the routing table of the host. The softlink device of the router through which we wanted to ping had to be an entry in the routing table. This meant to change the routing table of the host for every ping command a user executed. For several concurrent users we would have had to manage the access to the routing table of the host. But microvar was then

updated by its author with two Loadable Objects traceroute and ping. Now it is possible to load a ping object it the VR and the command is executed from within the VR itself.

Chapter 4

User View

4.1 Introduction

The user view of the application consists of mainly four pages, the login page, the create network page, the select router page and the configure router page. The user logs in with her user name and password. If the password is correct, the user is assigned a user-ID. At the first login the user receives a new user-ID which is then saved together with the user name. On the next login the user receives the same user-ID. A user with a valid password is then taken to the next page, the create network page. If the user already has created a network during a former session, this topology gets displayed, else the user can create a new topology. When the topology has been created or changed, the user may go to the next page, which will also start the routers. On this page the user can click on the router icon representing the router she wants to configure. When she has done this, a new page opens, where she can enter commands to the router. Commands like router configuration commands and the ping command can be entered. After she has finished configuring the router she may go back to the select router page to pick another router or she may leave the site. By default the network topology and the router configuration is saved. To make changes to the network topology, the user can go back to the create network page. See figure 4.1 for an overview. See figure 4.2 for the create network -, figure 4.4 for the select - and figure 4.6 for the configure page.

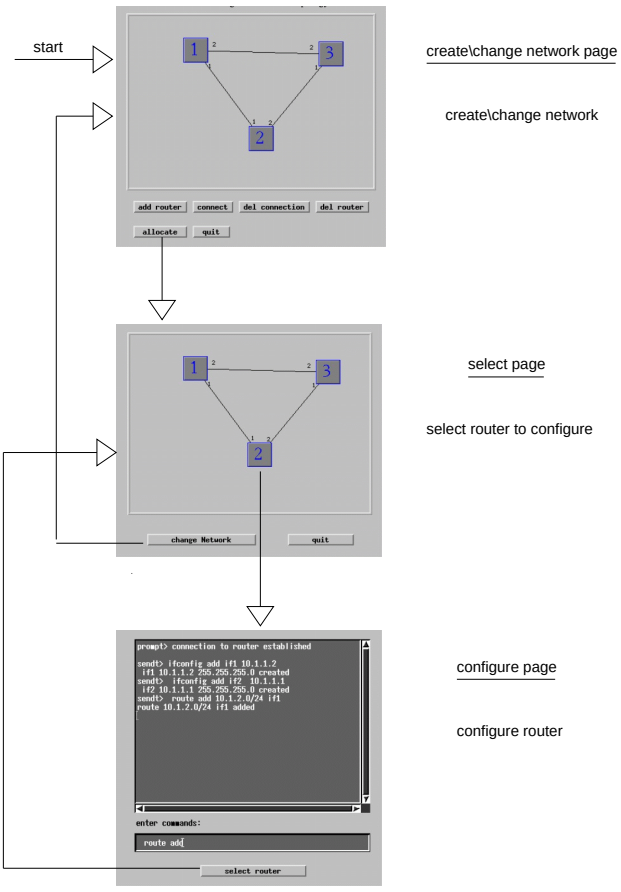

Figure 4.1: Application overview

4.2 Functions of the Create Network Page

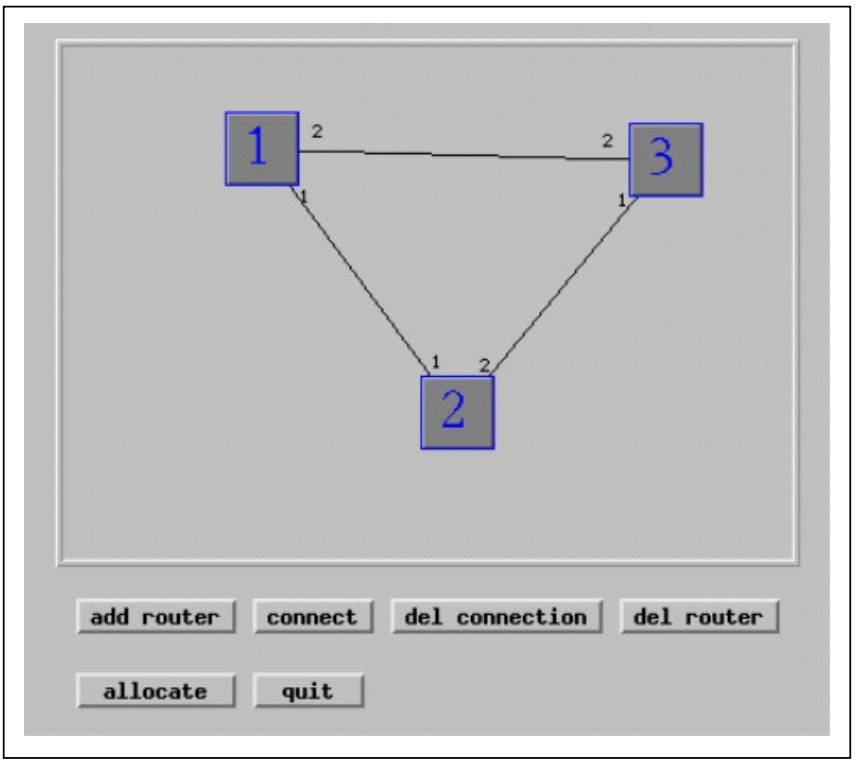

Figure 4.2: Create\change network applet

Element	Functionality
`add router`	A new router is created.
`connect`	Two routers are connected. For each connection an interface is created on each side of the connection.An interface has a natural number as label. The number gets incremented by one for each new interface.(If interfaces were deleted on this router before, the numbers that those interfaces had, are used first)
`delete connection`	A connection between two routers is removed and interfaces attached to the connection are also removed. To remove the connection the interfaces should not be configured.
`delete router`	Routers are deleted. Deleting a router also deletes the connections to other routers. To delete the connections, the same restrictions apply as in the function `delete connection`
`allocate`	Changes to the network topology are applied and the user is taken to the select page.
`quit`	Changes are saved and the allocated resources are freed.

4.3 Functions of the Select Page

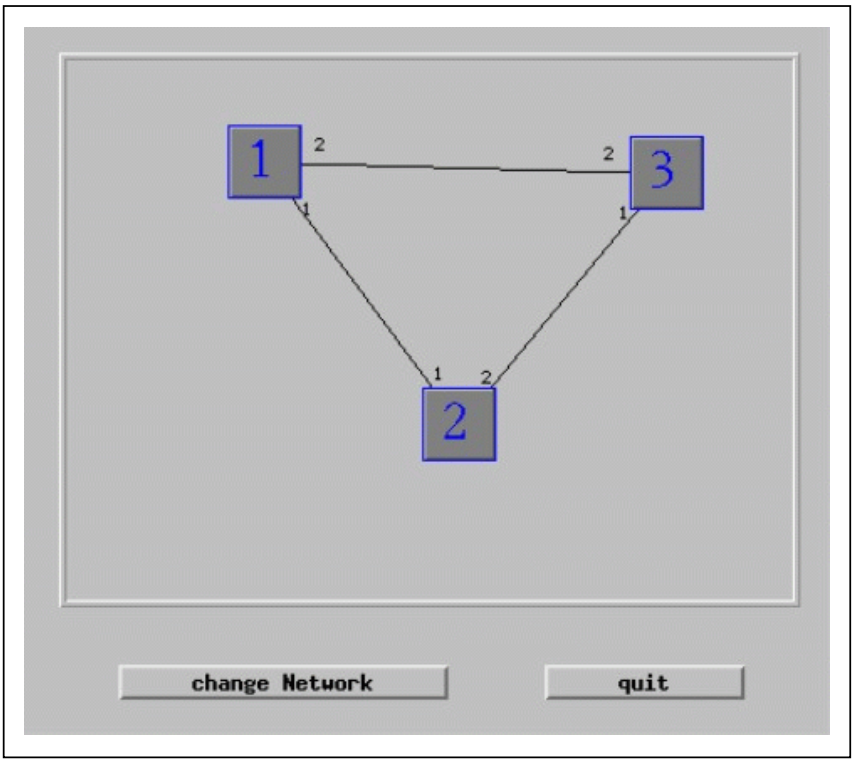

Figure 4.4: Select applet

Element	Functionality
`router icon`	Opens the configure router page.
`change network`	Opens the create\network page.
`quit`	Changes are saved and the allocated resources are freed.

Figure 4.5: Select functions

4.4 Functions of the Configure Page

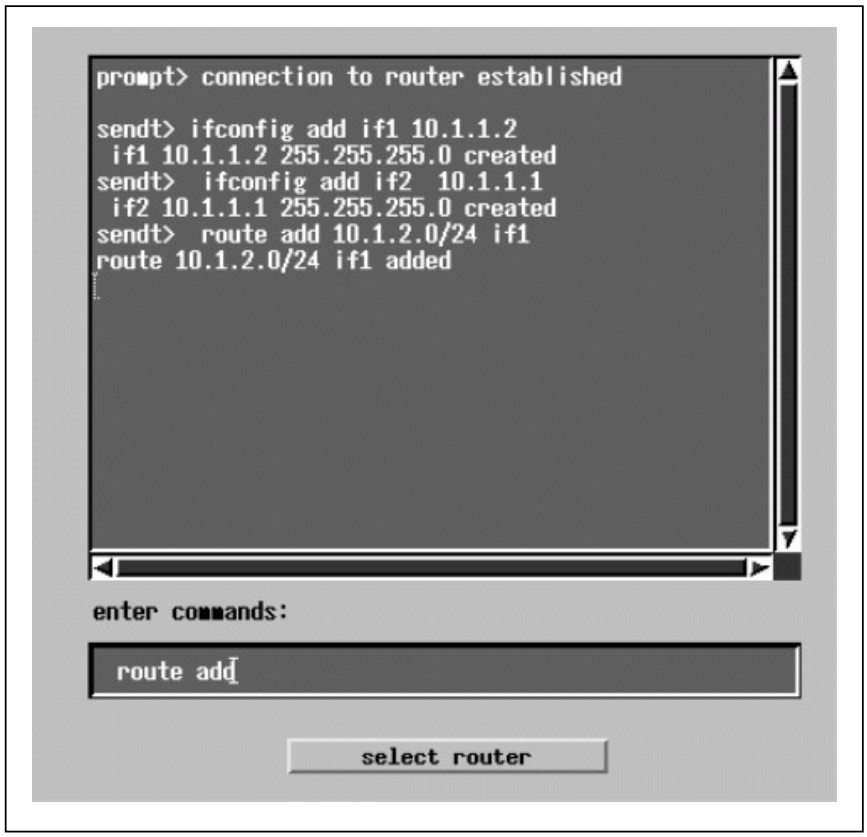

Figure 4.6: Configure applet

Element	Functionality
`select router`	Reopens the select page.
`interface`	The interface is used to send the commands to the routers

Figure 4.7: Configure functions

4.5 Router Commands

The following commands can be sent to a virtual router:

configure interface

```
ifconfig
ifconfig if<number>
ifconfig add if<number> <ip-address> [<network mask>]
ifconfig if<number> ip <ip-address>
ifconfig if<number> nm <network mask>
ifconfig if<number> delete
```

route

```
route
route add default if<number>
route add <ip-address> if<number>
route add <ip-address></><bitcount mask> if<number>
route del <ip-address>
```

test

```
ping <ip-address>
traceroute <ip-address>
```

general

`close` (This command closes the connection to the router explicitly.)

Chapter 5

Design

5.1 Overview

In this chapter we present the design of the wivrec application. First we will present the architecture of the application in an overview, then we describe the components of the architecture in detail.

Used Programming Languages and Tools

The web pages are programmed with php [PHP] and Java [Java].For the Java part no swing components[Hor00][Javb] were used and the Java version for classes used in the applets is 1.1.8. The applets are running well in browsers supporting Java 1.1.5. The Java classes not used in applets are sometimes of Java version 1.2.2. The web pages are installed on an apache http server[Apa] on Linux[Lin]. Microvar is installed on the same host as the web server. In this thesis we will mostly use the UML notation to describe design components and class - or object relations.

5.2 Architecture

For an overview of the architecture see figure 5.1.

The wivrec application has two components, the first component consists of the web pages , they are the interface to the user. The second component is a stand alone Java program, named administrator, which executes commands from the applets, stores application data and administers the router processes. By administering the VR's, the administrator interacts with the microvar component. We will give a short overview of the web pages and the administrator, a more complete description of both will follow in the next sections. The microvar component has already been presented in chapter two. Those topics

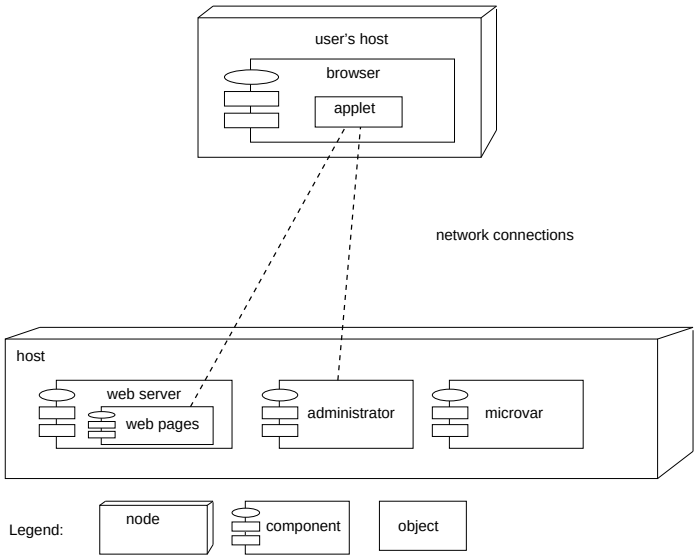

Figure 5.1: Architecture overview

that concern both parts, like data administration and applet - administrator communication, are presented in the web pages section.

5.2.1 Introduction Web Pages

The web pages are dynamic web pages, written in the language php. They reside on a web server, it could be any web server which contains a php module, and are viewed by the browser of the user's host. To receive the web pages, the user has to type an URL into the browser, as with any other web page. The user will get the HTML pages which were generated by the php scripts, that contain the Java applets. The applets have been presented from the user point of view in chapter `user view of the Wivrec application`. The Java applets, while running on the user host, can open TCP connections to the administrator component.

5.2.2 Introduction Administrator

The duties of the administrator program are to manage the connections to the applets, execute the commands from the applets and controlling the router processes. The administrator program has to run as long as the web pages are available and it has to run on the same host as the web server, since an applet can only open a TCP connection to a host it is loaded from [Javb].

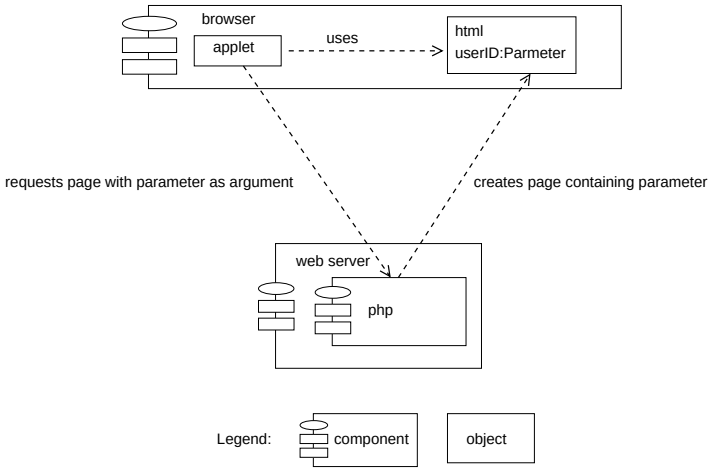

Figure 5.2: Data flow between pages

5.3 Web Pages

5.3.1 Overwiew

First we will describe the construction of the web pages and how information is transmitted between them. Further we look at how data is transmitted between an applet and the administrator and how commands are sent to the administrator. As next we will give an introduction to the data administration and then we will look at those topics for each applet separately.

5.3.2 Page Construction

The web pages are HTML[HTM] pages generated with php. The HTML pages contain Java applets as embedded objects[Javb]. We do have mainly five pages, those are the login, user, create\change network and configure pages. The login and user pages are for administrative purposes. The other pages contain the functionality of the client side of this application, here we concentrate on those latter pages. From the login page we get to the user page and from there to the change network page. Then we get to the select page and then to the configure page.
The order is the same in the other direction:-configure-select-change network.
For an overview see figure 5.3

The pages have to convey information to each other. When one page opens another page, some information has to be transmitted to this page. For an overview see figure 5.2.

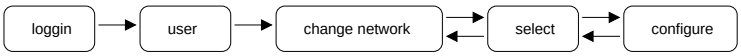

Figure 5.3: Sequence of pages

5.3.3 Data Administration

The data of the application consists of the topology the user has created. The topology information includes the network topology and the configuration information of a single router.(Internally the topology information is represented as Java objects and their relations.) Because of security restrictions on the applet side it is not possible to save data to disk on the client side[Javb], as has already be mentioned in 5.2.2. The data is sent over a TCP connection (see section 5.3.4) to the administrator where it gets saved. When a page opens, it retrieves the information from the administrator and before the page is closed, the possibly changed information is sent back to the administrator, where it gets saved. The user works on one page at a time, this ensures that the data remains consistent.

5.3.4 Applet - Administrator Communication

In this section we describe the communication between an applet and the administrator program in general.

The communication between an applet and the administrator is implemented using TCP connections. The administrator acts as a server and is waiting for connections on a predefined [1] port.

The applet acts as a client and connects to the administrator. When a connection request arrives to the administrator, the administrator creates an Object that handles the connection and continues to listen for other connections. See figure 5.4

A `net action` is a message exchange sequence between an applet and the administrator, containing the command or request from the applet to the administrator and optionally containing the response from the administrator. The different net actions are described in 5.3.5
A new TCP connection is used for every net action. Before starting a net action the applet connects to the administrator and afterwards the connection is closed again. The net actions for one applet are executed sequentially and thus also the TCP connections are opened and closed sequentially. The format of the messages are plain text ASCII Stings or byte streams for transmitted data. The main message String for all net actions is action-command 5.3.4, which contains the command or request from an applet to the administrator.

[1] predefined by the network administrator hosting the server

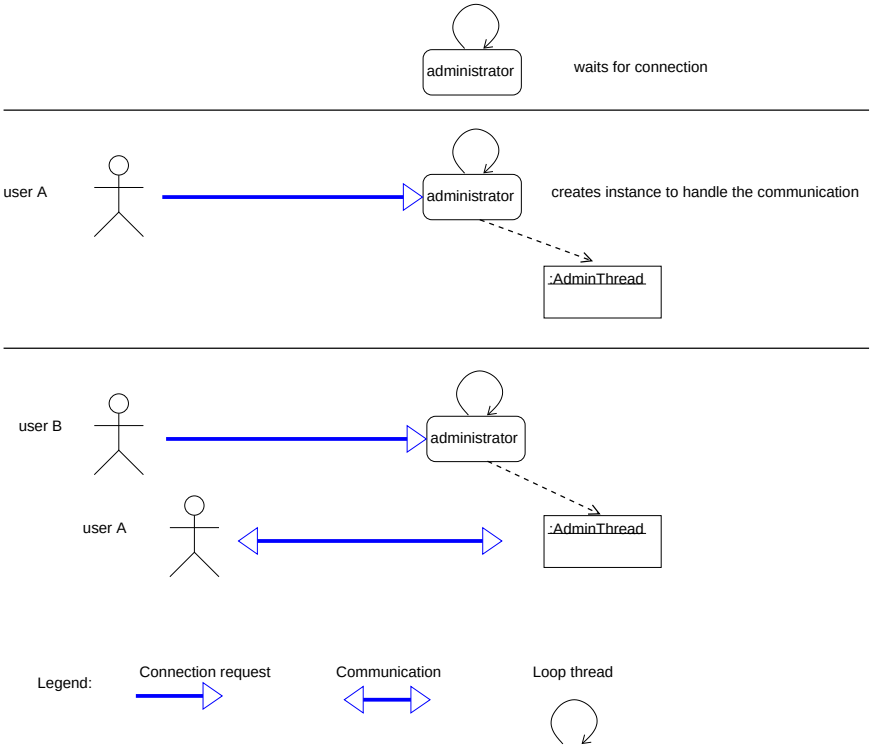

Figure 5.4: Connection handling of the administrator

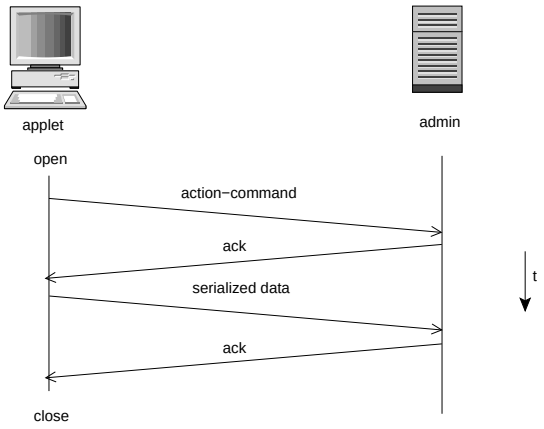

Figure 5.5: Message exchange sequence for the save command

Action - Command

The action-command is a String containing parameter-value pairs. The first parameter is the word“userID” followed by a word containing the value. The second parameter is the word “use” followed by a word describing the action. Parameter and value are separated by a blank space, as are the parameter-value pairs.

5.3.5 Net Actions

Save Action

semantics:
the save action command is used to send data to the administrator where it gets saved to a file.
message sequence:
an action command is sent to the administrator. The administrator sends back an ack message. Save action sends the data, admin [2] returns an ack.
Figure 5.5 shows the message exchange sequence.

Allocate Action

semantics:
Admin gets the command to allocate the resources, which updates the processes. Router processes are started for routers that the user added to the topology and router processes are shutdown for routers that the user deleted. message sequence:
The applet sends the message action-command.The administrator sends back a

[2] in this chapter, `admin` will often be used as a shorthand for administrator

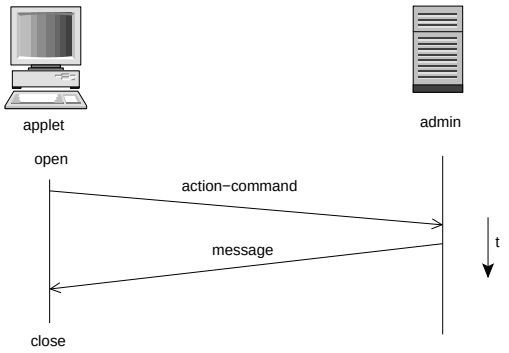

Figure 5.6: Message exchange sequence for the allocate command

message, informing if the command was successful and closes the connection. Figure 5.6 shows the message exchange sequence.

Retrieve Action

semantics:
the applet requests the topology data. If the user had already created the topology it gets send, and the applet displays it, else nothing gets displayed. message sequence:
The applet sends the command-action to the administrator. The administrator sends 0 if there is data to send. The applet sends ack. Admin sends the data, the applet responds ack. If there was no data to send, admin sends -1 and the applet responses with ack.
Figure 5.7 shows the message exchange sequence.

Quit Action

semantics:
the router processes are shut down when the user closes the session. message sequence:
The applet sends the command-action to admin.
Figure 5.8 shows the message exchange sequence.

5.3.6 Page Structure

Create\Change Network Page

When the change network page is opened the data containing the topology information has to be retrieved. The applet retrieves the topology data with the retrieve action. On the CHANGE NETWORK PAGE the user changes the network by adding or deleting routers. This changes have to be updated in the administrator. This is done with the save command. When the user leaves the

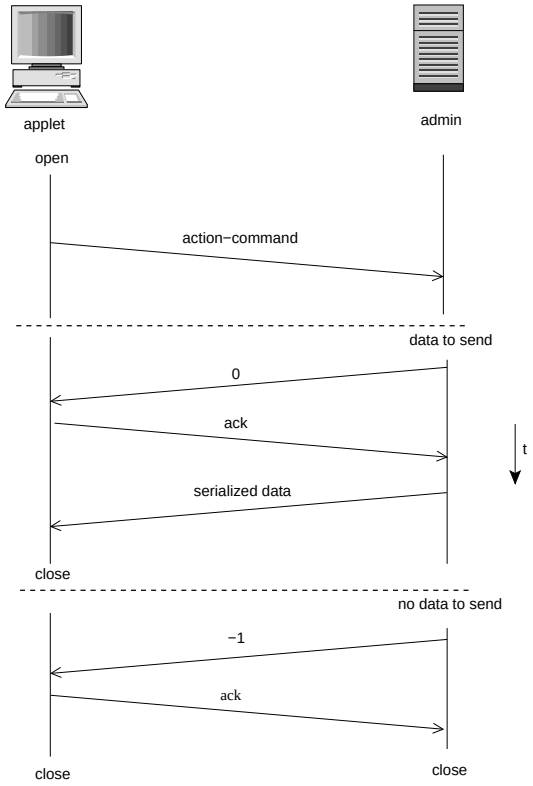

Figure 5.7: Message exchange sequence for the retrieve command

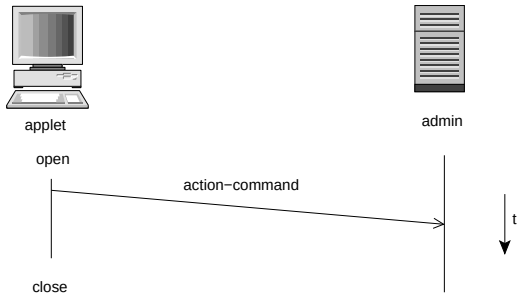

Figure 5.8: Message exchange sequence for the quit command

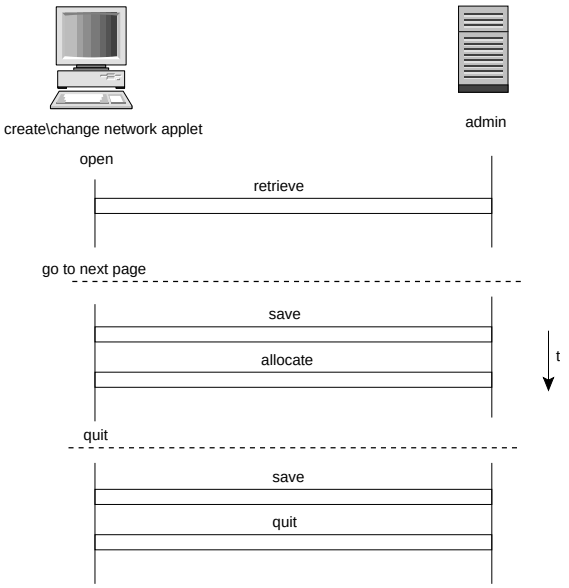

Figure 5.9: Connection sequence for the change network page

page and goes to the select page the data is sent to admin for admin to update the data. Allocate updates the router processes by starting them for added routers or stopping them for deleted routers.

Then the select page is opened. If the user quits the page the save action and then the quit action are used.

For the connection sequence of the page see figure 5.9
For the data flow overview see fig. 5.10

Select Page

The applet retrieves the topology data with the retrieve action command. The topology gets displayed. On this page the user may move the routers as sole topology changes. The user may click on a router to configure the router. Or the user may terminate the session with quit or she may reopen the create network page. In each of this cases the first step is to store the data with the save action command. If the user quits the quit action command is used. For the connection sequence see fig. 5.11
The data flow is the same as for the change network page.

Configure Page

On this applet no topology is displayed, the applet serves as an interface to the router processes. When this applet starts it connects to admin. The user input

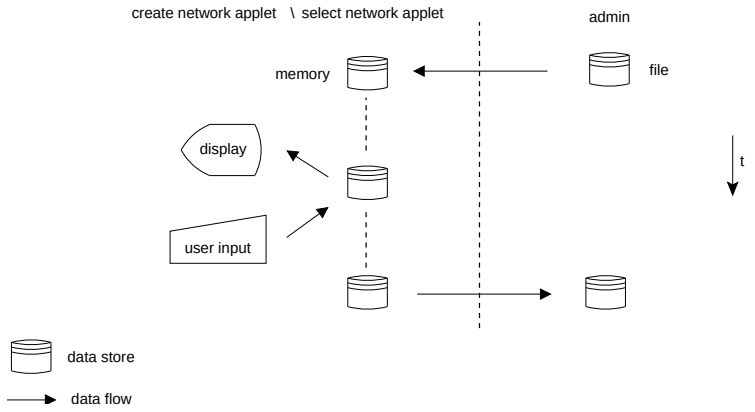

Figure 5.10: Data flow overview for the change network page and the select network page

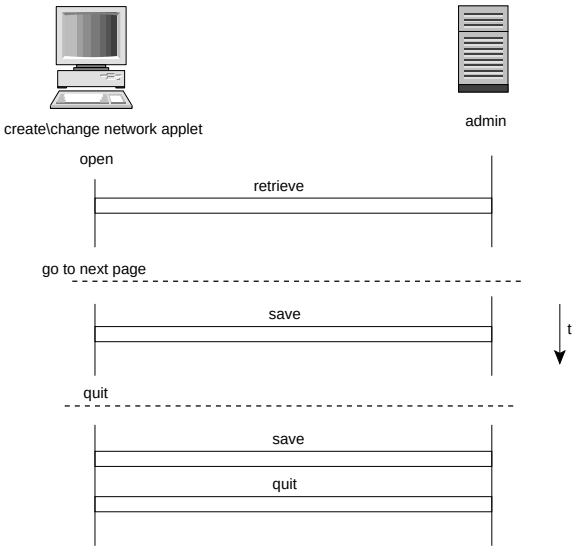

Figure 5.11: Connection sequence for the select page

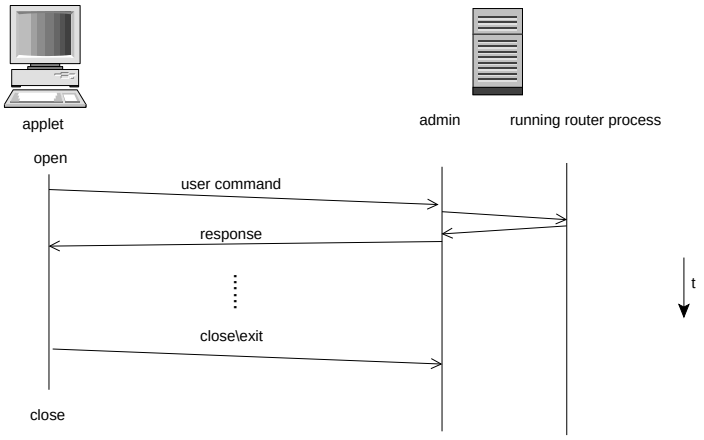

Figure 5.12: Message exchange sequences for the configure network page

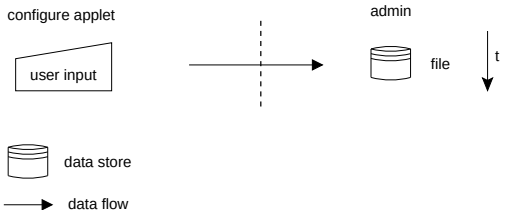

Figure 5.13: Data flow sequence for the configure network page

is then sent to admin which transfers it to the running router. The administrator sends the response from the router process back to this applet. This goes on until the user sends the close command or the page is left. If the page is left the applet itself sends the close command. For the message exchange sequences see figure 5.12
For a data flow overview see figure 5.13

5.4 Administrator

5.4.1 Overview

In this section we present the second part of the application, the administrator program. Admin receives commands or requests from applets and executes the appropriate commands. For an overview of the commands see figure 5.14. For every net-action (see 5.3.5) there is a counterpart in the administrator which fulfills the request or executes the command. In the first part of this section we describe the semantics of the network topology data and the interrelation of the network topology data and the router processes. In the second part we present each command separately. We will focus especially on the router configuration.

Network Topology Information

The web pages and the program administrator both share the same data. The data is sent from the web pages to the administrator and vice-versa. The data represents network topology information and router configuration information.

The network topology information is represented by router objects and connection objects. The router configuration information is also represented with object entities. E. g. for a configured interface the router has an object interface and for a routing table the object router has an object routing table.
On the implementation level these routers and connections are implemented as classes. For a thorough description of the data at the implementation level, see the chapter `Implementation`. The aim of the data model is to represent the router processes, thus the components of the router processes are represented as objects in the data model.

Defining the Terms `Topology-view` and `Process-view`

In this thesis we sometimes use similar expressions to describe routers for both (running) router processes and routers as elements from the data structure. For example configuring an interface might refer to an interface of a router process or an interface of a router in the topology [3]. To avoid any confusion we use the terms `topology-view` when the context is the data structure and `process-view` when the context is router processes. Those definitions will also be useful to explain some important aspects about the consistency of the data and the mapping between the topology and the VR's. For an overview see Figure 5.15.

[3] in this chapter `topology` is a shorthand for *data elements representing the network topology* and *network topology information*

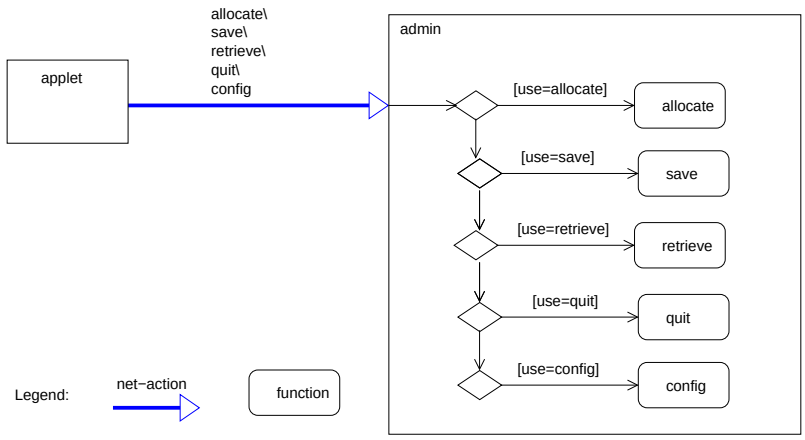

Figure 5.14: Administrator functions

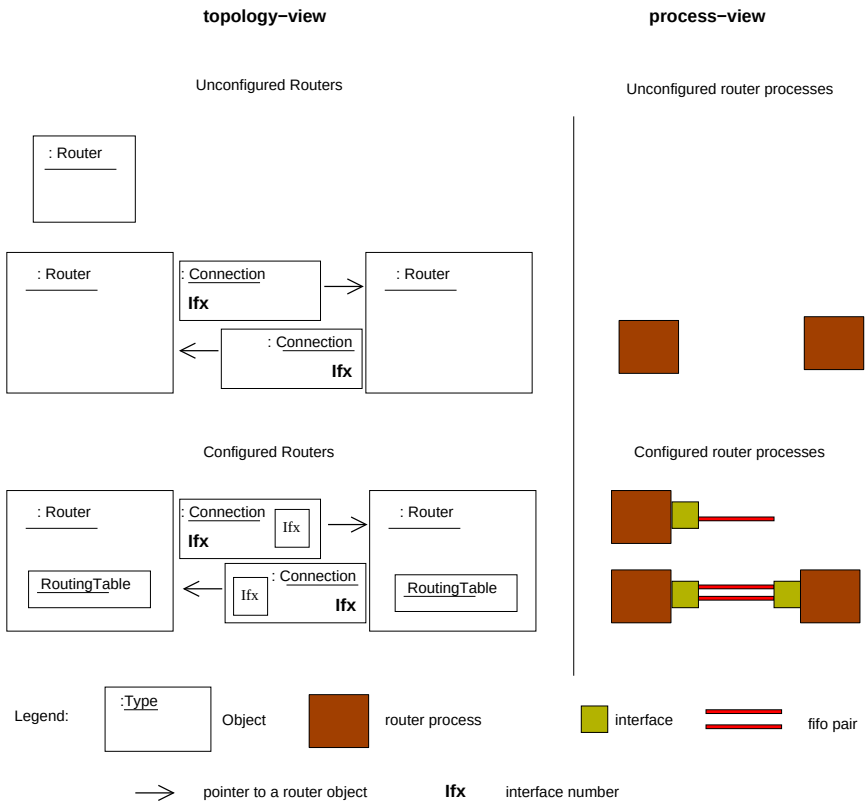

Figure 5.15: Overview topology-view and process-view

5.4.2 Configuring the Routers

Overview

In this subsection we will see how a router process is being configured. Routers are configured by sending them configuration commands. Not all commands are entered by the user itself, some are directly sent to the router process by the administrator. To send a command the `administrator-router protocol` is used, which is described in the next paragraph. We will distinguish between `user commands` and `microvar commands`. Both sorts of commands are described in the following paragraphs.

Next we will look at two user commands, one for adding an interface to a router and the other for deleting an interface from a router. In these examples we can see how a user command may trigger several microvar commands. To understand the command for adding an interface we have to recall how a microvar router is connected, this will be described shortly in paragraph Connecting two Routers on the same Host.
Then we will explain why we need to keep track of the router process configuration in Configuration State of a Router Process. And next we will describe in the subsection Configuring an Interface and Updating the Configuration State how the configuration information on both the topology-view and the process-view can be updated, so that all the configuration information from a router process is contained in the topology.

Administrator-Router Protocol

The administrator maintains the TCP connection to the router process as long as the router is running. The core of this protocol consists of the router waiting for a command, administrator sending that command, administrator waiting for the response and the router sending the response.

User Commands

All existing commands to a microvar router are listed in the microvar manual, for example in [FB]. The user commands are a subset of those commands, they are the commands a user sends directly to a router, they are listed in chapter `user view of the wivrec application`

Microvar Commands

The microvar commands are mostly coupled to user commands, or they are sent to the router independently, they are the commands to a microvar router which the user of the application doesn't need to know (hidden commands). In this paragraph we describe the microvar commands used to configure an interface of a router process. For an overview of the microvar commands to add an interface see figure 5.4.2

Attaching an Interface to a router process

`ifconfig if<number> qs create droptail`
`ifconfig if<number> qs chain 0 1`
`ifconfig if<number> qs chain 1 0`

Attaching a Fifo to an Interface
`ifconfig if<number> connect fifo <fifonr> <r|l>`
Disconnecting two Routers
`ifconfig if<number> disconnect`

Figure 5.16: Microvar commands to add an interface to a router

Connecting two Routers on the same Host

A connection between two router processes is established with two fifos. In `microvar` a user has to start a script with the amount of desired fifo pairs as argument[4]. The fifos are named `link<nr>l\r`. One fifo is attached to one *router interface* and the other fifo of the pair is attached to the *router interface* of the other router. An important point to notice is that the routers need interfaces to get connected.

Configuring an Interface

When the user configures an interface of a router process with the user command `ifconfig add if<number> <ip> [<nm>]` the instance which is handling this user command knows which two routers will be connected over this interface. This is known, because the user has connected the router in the topology-view in the CREATE NETWORK PAGE. The handling instance sends the user command to the router process, using the administrator-router protocol. Then the microvar commands for attaching an interface are sent to the router process. The microvar commands for connecting two routers are coupled to the user command for configuring an interface. It may seem better to connect the `routers` first, since this connection models a physical connection of real routers, but it is not possible to connect two `routers` before they have an interface. Therefore they have to be connected after they have received an interface. This will be explained more thoroughly in paragraph `Configuring an Interface and Updating the Configuration State`
See figure 5.17 for the message exchange sequence.

Deleting an Interface

For deleting a router all connections to other routers have to be removed. The interfaces have to be removed from both sides of the connection. The user re-

[4]in the administrator program the fifos are created on demand

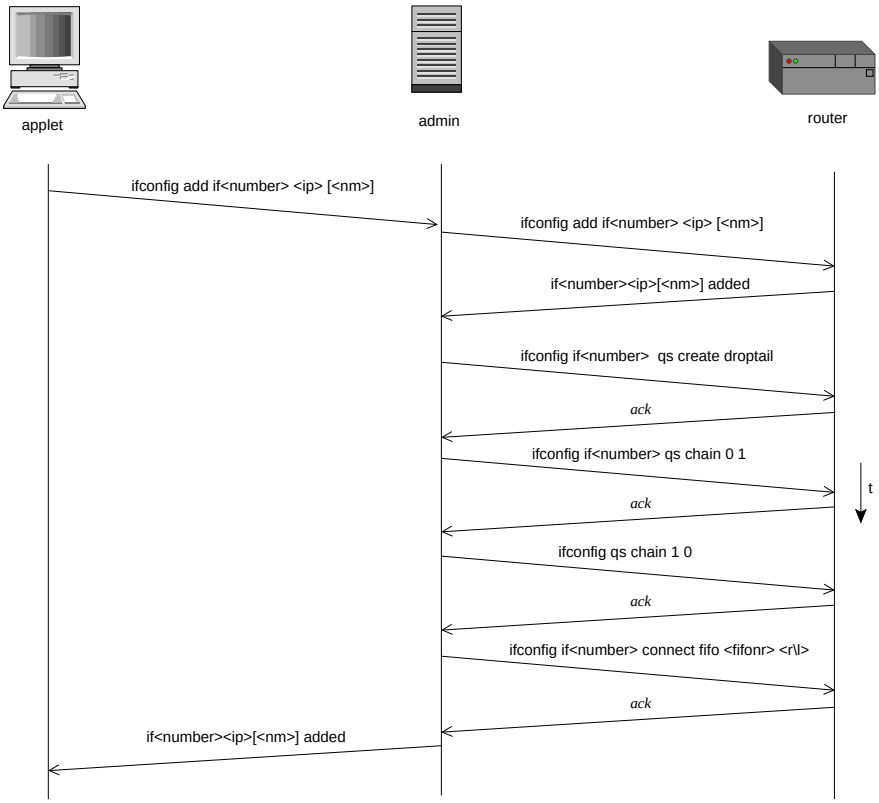

Figure 5.17: Message exchange sequence for adding an interface

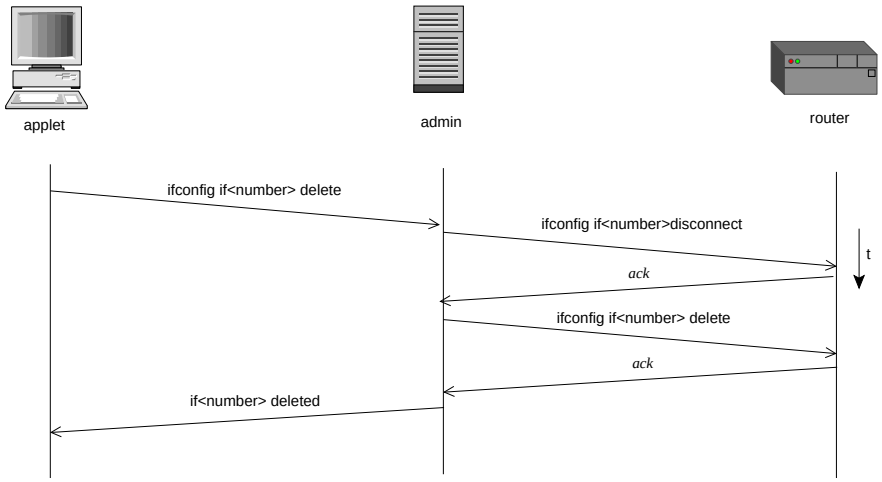

Figure 5.18: Message exchange sequence for removing an interface

moves the interface with the user command
`ifconfig if<number> delete`
See figure 5.18 for the message exchange sequence.

Configuration State of a Router Process

A running router process has a configuration state but this state is not entirely accessible from the outside. We may query a router about its configured interfaces or about routing entries, but we can't find out to which router it is connected and which fifos are used for this connection.

Configuring an Interface and Updating the Configuration State

For configuring an interface on a router process, configuration information of the corresponding router from the topology-view is used. In the CREATE\CHANGE NETWORK PAGE the routers in the topology-view are connected by the user and an interface number is assigned (automatically) to each connection. When we configure the interfaces of the router processes we need this connection information, which is contained in the topology-view. The microvar commands to connect router processes are transparent to the user, the connection information has to be taken from the topology-view. Therefore, a first step in adding an interface to a router process is to ensure that this interface exists in the topology-view. When the microvar commands to connect the router processes will be executed, this will allow to get the connection information from the topology. Figure 5.19 depicts the different configuration states of the topology-view and the process-view when an interface gets configured on two router processes

to be connected.
In step one of the figure we see that the corresponding router from the topology-view has an interface and a connection object with the information to which router it will be connected.

Once the handling instance knows that it is allowed to add the interface to the router process, the command to add the interface is sent to the router, as described in the paragraph `adding an interface`.

When an interface is added to the router process the IP number and possibly the net mask is configured at the same time. This configuration information is saved in the topology-view. In this step we update the topology-view with the configuration state of the router process.

Next a fifo is attached to the interface of a router process, which is the configuration step to connect the router processes. To connect a fifo unit to an interface of a router process, the handling instance would query if the router it will be connected to has already a configured interface and thus an attached fifo unit. If this is not the case, as in figure 5.19 in step 2, the handling instance would create a new fifo pair and attach one fifo of the pair to the interface. The corresponding router in the topology-view gets updated with the fifo number and the fifo direction.

If the user configures an interface of a router process, and the router it will connect to has already a configured interface and thus an attached fifo, the matching fifo to this fifo gets attached to the router the user configures. Then the topology gets updated with the information about the used fifo, as can be seen on the figure 5.19 in step 4 and 5.

5.4.3 Command Allocate

The administrator maintains the handles to the running processes for each user which has allocated (running) routers. The administrator receives the allocate command when a user leaves the CHANGE NETWORK PAGE. Then the router processes have to be updated. Processes of deleted routers have to be stopped and processes for newly created routers in the topology-view have to be started. [5] For an overview see figure 5.20. The subcommand for starting a router is explained in the next paragraph.

Starting Routers

In allocate a list of all routes is extracted from the topology. For the routers [6] that don't have a matching process [7] a router process gets started. A script file

[5] We may also view allocate as a function mapping the topology- to the microvar view.
[6] topology-view
[7] process-view

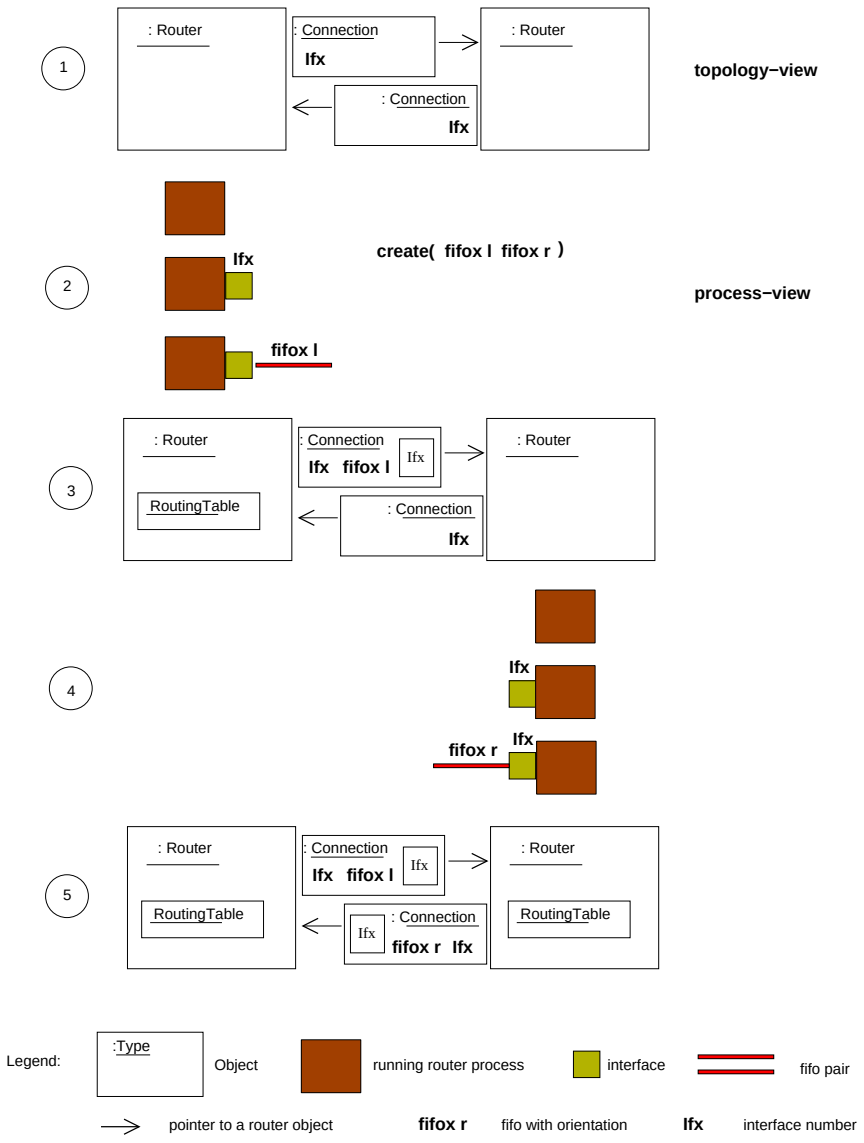

Figure 5.19: Adding an interface

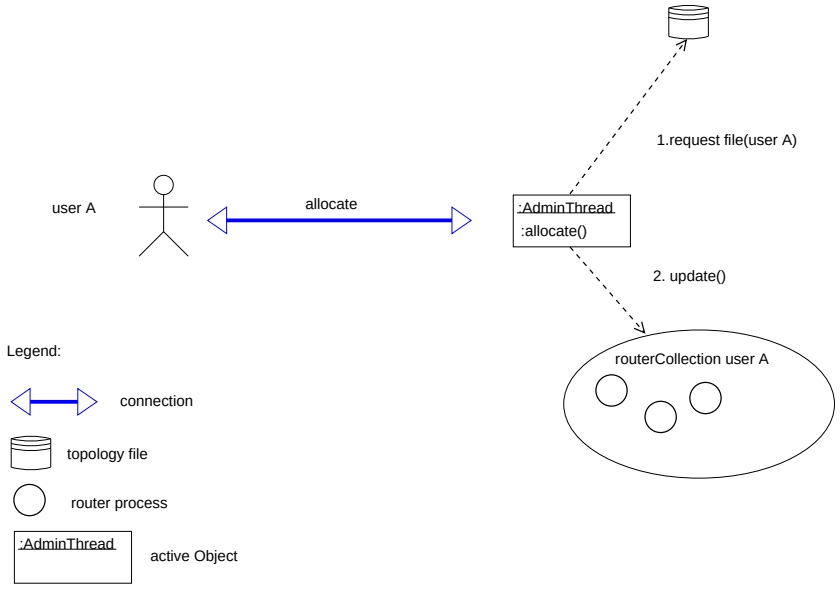

Figure 5.20: Allocating router processes

is created with the configuration information from the router of the topology-view, and this script is the configuration file for the router. The router reads this file at first as configuration file. If the router from the topology-view was not configured, the script would be just empty.

5.4.4 Saving the Topology

During a save-action message exchange sequence admin receives the network topology information from the applet which has initiated the net-action. This network topology information is stored in a separate binary file for each user.

5.4.5 Retrieving the Topology

Upon receiving the retrieve command the administrator reads the file containing the network topology information and sends it to the applet requesting it.

5.4.6 Shutdown the Routers

After receiving the `quit` command the administrator sends the microvar-command shutdown to the router process. The router process stops executing and releases its resources after receiving this command. Notice that this is a case where a microvar-command is directly sent to the router process. Here the communication with the router was not initiated by a user-command.

Chapter 6

Implementation

6.1 Web Pages

6.1.1 Overview Packages

The classes from the client side are divided into the packages: applet, controls, net, topology, action. The applet package contains the applet classes and the classes to draw the applet, like coordinate area. The controls package could be a sub package of the applet package, for it contains mainly the classes to represent the buttons on the user interface. The package topology has the classes representing the topology and maintaining the router states. The Objects of those classes are serialized and de serialized on the server side and vice versa, therefore this package contains the same classes as the topology package on the server side contains. The package net is used mainly for the TCP connections. The package action has classes representing actions such as connecting routers, opening a new applet, or sending a command or request over the net(net action).

6.1.2 Overview Applets

the applets CreateApplet, SelectApplet, ConfigureApplet are interfaces to the user, thus displaying data to the user and accept data from the user. They also take the client part in the client server communication.

Initializing the Topology at Applet Start

example createApplet: the applet gets loaded from the browser.before it gets completely displayed, it opens a connection to the server and requests the topology of the network. if the user had already created the topology it gets send, and the applet displays it, else nothing gets displayed. The user can then press the buttons and thus trigger different actions. For example adding a router.

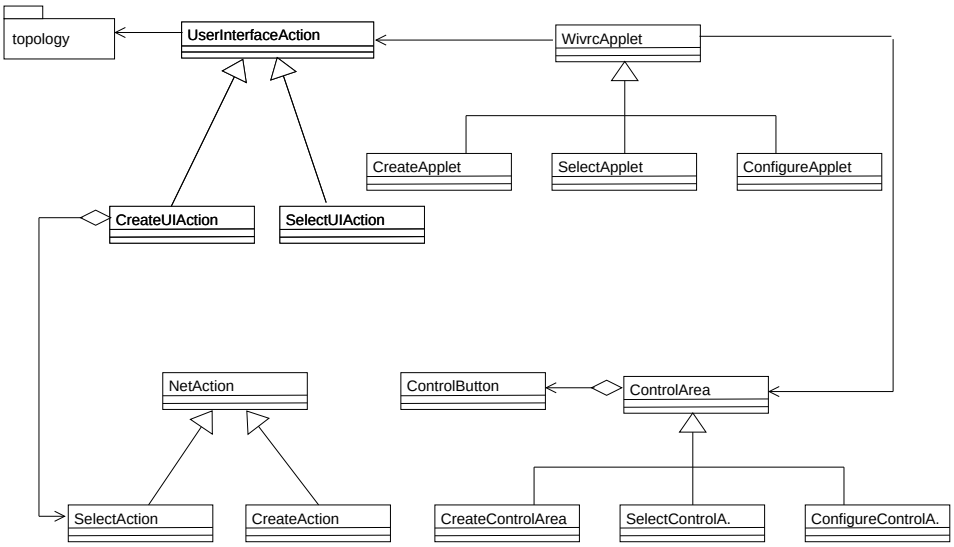

Figure 6.1: Classes overview

6.1.3 Classes

An appropriate class to start with is the class `WivrcApplet`, which is the superclass of the classes `CreateApplet`,`SelectApplet` and `ConfigureApplet`.The classes SelectApplet and CreateApplet are more related to each other than the class ConfigureApplet. The superclass has an instance of the classes CoordinateArea,ControlArea,and FramedArea, those classes are extensions of the classes Java.awt.Panel or Java.awt.Canvas which are classes from the standard Java API [Java]. Those classes are used for the graphical representation. CoordinateArea is the canvas where the routers are displayed on.The canvas listens to mouse Events and notifies the Applet if such an event occurred. The applet delegates the event handling to an instance of a subclass of the class UserInterfaceAction.
At creation time of CreateApplet the instance variable of the superclass userInterfaceAction is instantiated with an instance of its subclass CreateUIAction. CreateUIAction is responsible for handling the events. The same is true for the class SelectUIAction which is responsible for handling the events for the SelectApplet applet. ControlArea is the superclass of the classes CreateControlArea and SelectControlArea. ControlControlArea displays the buttons of the user interface for the CreateApplet and also forwards mouse events to the WivrcApplet class which in turn forwards them to CreateUIAction class.
The instances of the UIAction[1] classes get notifications of events triggered by pushed buttons or they get notifications about which of the displayed routers

[1]this is a shortcut for CreateUIAction and SelectUIAction, it follows from the context, which class is meant

Figure 6.2: Initializing the topology at applet start

was selected with the mouse device. The UIaction classes have to react on mainly two different ways, one is to perform an action on behalf of which button was pressed the other on which router was selected after a button was pressed. For example when a router was moved the canvas has to be repainted. In this case the UIAction instance sends a message to the controller (controller is the name of the actual applet instance), that the canvas has to be repainted. The controller in turn calls a method of the UIAction instance. The method for painting the router representations is implemented on the UIAction side because the routers are painted differently for different states of UIAction. If for example the routers have to be repainted because of a deleted connection they are painted differently as if when they have to be repainted because of a move action(change location on canvas).
The actions that are taken because of pushed buttons are mostly represented as separate classes. For example when the user wants to quit the applet he presses the quit button. An instance of QuitAction class is initialized. QuitAction, a subclass of NetAction, opens a TCP connection to admin and sends admin the command to close. Admin will then shut down the router processes. The classes for actions that have to open a TCP connection are all subclasses of NetAction. Another sort of actions are opening new applets. Those classes implement the interface PathIF. PathIF holds all the paths which the client side uses as String variables. This interface also has the port number of the port on which admin is listening. A third kind of actions are the actions taken to manage the topology. For example connectAction, an instance of ConnectAction class, registers which two routers have been clicked on after the connect button had been pressed. After that it sends a message to the actual instance of class TopologyImp,which connects the routers. The package topology is also used on the server side and will be explained in the next chapter.

6.2 Administrator

6.2.1 Overview

In this section we present the second part of the application, the administrator program. As we have seen in the chapter describing the design of the administrator, the administrator has to respond to requests or execute commands dispatched form the web pages. For every net-action presented in chapter design, there exists a counterpart in administrator. The size of the code for executing the commands differs considerably. The code for executing a quit action command is implemented in a simple method of class Administrator and uses one class, whereas the code for executing an allocate command encompasses several classes, which are grouped in package allocate. The code for saving or retrieving data(save action) is mainly found in package topology, the code for configur-

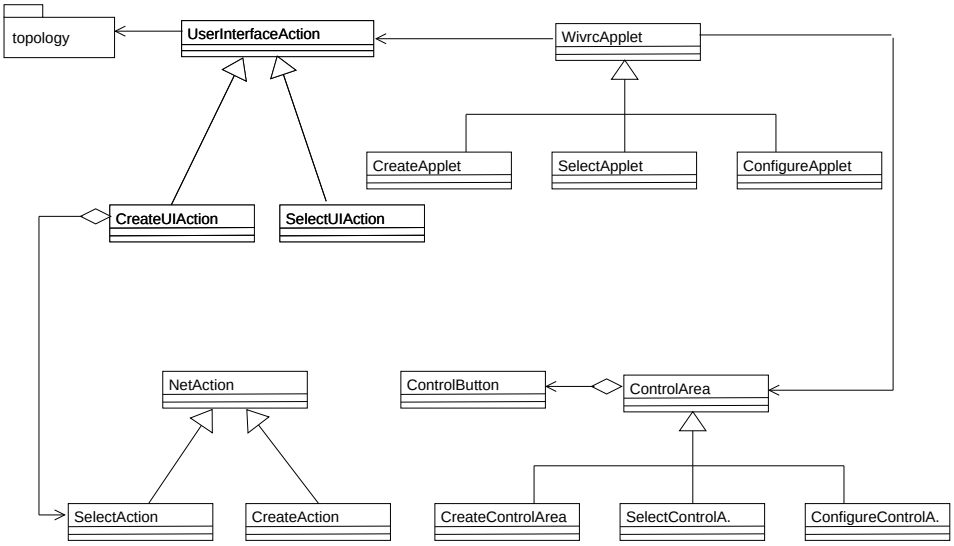

Figure 6.3: Config classes overview

ing a router is contained in package config. The classes from the administrator program are divided into the packages `config` `administrator` `topology` and `allocate`. The topology package contains classes to represent and manipulate the data representing the network topology.

6.2.2 Package Config

The classes from package `config` manage the communication with the routers.

Dialog Class

When admin executes the config function, administrator[2] creates a `dialog` object. The start method of dialog is called and dialog starts a loop. At the beginning of the loop it waits for a command from the user. Once the command of type String arrives the first token of command gets analyzed by a formerly initialized instance of the class StringAnalyzer(st). In StringAnalyzer a constant is declared for each valid token. When a valid token is recognized, `st` returns the constant to dialog. If the token is not valid, the constant `INVALID_TOKEN` is returned to dialog. Dialog sends a syntax error message to the user. Else the constant gets analyzed by dialog. As many tokens as necessary are analyzed to identify the command. When the command is identified, the command gets executed. There is to say that not all tokens from the user command are analyzed

[2] AdminThread, an inner class of class Administrator

at this level. Command arguments like IP addresses are parsed by the microvar router itself.

the execution of the command yields a message which is returned to the user. dialog continues with the execution of the loop until it gets the CLOSE command from the user.before dialog is closed the changed topology is saved to file.

Command Classes

There are mainly three kinds of user commands. The first kind is a command of type ConfigCommand. Such a command changes the configuration of the router process. The topology has to be updated accordingly. For every command there exits an own class. The classes of type ConfigCommand extend the abstract class `ConfigCommand`. ConfigCommand is abstract because it is never used directly, only the subclasses are used. The second type of commands is the UserCommand. The classes of type UserCommand extend `UserCommand`. Commands of type UserCommand send a command to the router process, which doesn't change the state of the process. PingCommand is such a command. The third kind of command is a command to the administrator. Close is an example of such a command. This command ends the configuration session with the router, the command itself is not sent to the router.
The Command classes manage the dialog with the router process, implementing the administrator-router protocol 5.4.2. They may send a single String message and return the response to `Dialog` (for commands of type UserCommand) or they may send several messages, update the topology and then return the response. An example of a ConfigCommand is class `IfAdd`, which will be explained in detail.

Semantics of ConfigCommands

The user command is not completely analyzed in `dialog`, it is only analyzed as much as is necessary in order to decide which command it is and thus which type of object has to exchange messages with the router. Command parameters like IP addresses are not analyzed, because the command will be parsed entirely on the router side. The router answers with different messages. If there was a syntax error the router would return a corresponding message. If the sent command was correct, the router sends back a message that contains the updated settings. The settings are extracted from the message and the topology gets updated with those settings.

Example of a ConfigCommand class, class IfAdd

In dialog an instance of the class IfAdd gets initialized and its exec method is called. As parameters the method gets a reference to an object of type ConfigTopology, the router name, an instance of class administrator.RouterC, representing the connection to the router, and the command (`ifconfig add`

if<number> <ip> [<nm>]) as a String. Before sending this command to the router we must ensure that the router[3] has this interface.

```
//user isn't allowed to add this if
if(!configT.hasIf(ifx)){
    e.debug("try to add unconnected if");
    //send message to user
    return(error + "try to add unconnected if");
}
```

If the router in the topology-view has this interface, the command is then sent to the router.

```
//user is allowed to add this interface
//
//add interface to the router process
//
r.send(command);
```

The topology is updated with the new settings and the microvar commands for attaching an interface are sent to the router.

```
//update the topology with the net mask
configT.createIf(ifx,ips,nms);

//result is the return value of this method
result = " if" +ifx + " " + ips + " " + nms + " created";

//attach the interface to the router
r.send("ifconfig " + ifs + " qs create droptail");
String buffer;
buffer = r.get();
r.send("ifconfig " + ifs + " qs chain 0 1");
buffer = r.get();
r.send("ifconfig " + ifs + " qs chain 1 0");
buffer = r.get();
buffer = null; // the return value from the router isn't used
```

Then we have to make the connection between the router processes: For this we have to know if the interface of the other router process was already configured and in this case which fifo number and orientation it has.

```
int[]fifo = configT.getOtherFifo(ifx);
```

If the other interface was not configured we create a new fifo pair.

[3]the router in the topology representing this router process

```
if(fifo[0]== 0){         //no fifos for this connection created yet
    //get new fifo
    //int newFifo = 1;
    int newFifo =fifos.getFifo();
    fifo[0] = newFifo;
    fifo[1] = 0;           //int[1]={0=left;1=right}
      //create always left fifo first
```

Then we send the microvar command for attaching a fifo to the router

```
 r.send("ifconfig " + ifs + " connect fifo " + fifo[0] + " l");
```

We update the topology with the fifo information, and terminate the method by returning the result String to Dialog, which forwards it to the CONFIGURE PAGE

```
//update if with created fifo
configT.setFifo( ifx,new int[]{fifo[0],fifo[1]});
return(result);
```

For the entire code see A.3.1

6.2.3 Package Topology

The package `topology` contains classes representing the network topology and classes used to create, modify and update this topology. There are two main classes, the class Router which contains configuration information and the class Graph which maintains the network topology information. The classes Topology, DisplayTopology, ConfigTopology, CreateTopology and Serializer access the Graph class and update network topology changes. Some of those classes will be presented in detail later in this section.
The access classes to the Graph class all implement interfaces. Because of this the representation of the network topology in the application could be changed completely without changing the rest of the program code. For an overview [4] of the classes see 6.4 .

Graph

The class Graph is the core class of package `topology`. It is implemented as data structure undirected multi graph. The vertices represent the routers and the edges represent the router connections. The graph class is named GraphImp and implements the Interface Graph.
Methods of Graph:

[4] Each of this Classes implements an Interface and is named <interface name >Imp. Example: If the name of the interface is Topology, then the name of the class is TopologyImp. When the classes are used in the program they are referenced by their interfaces.

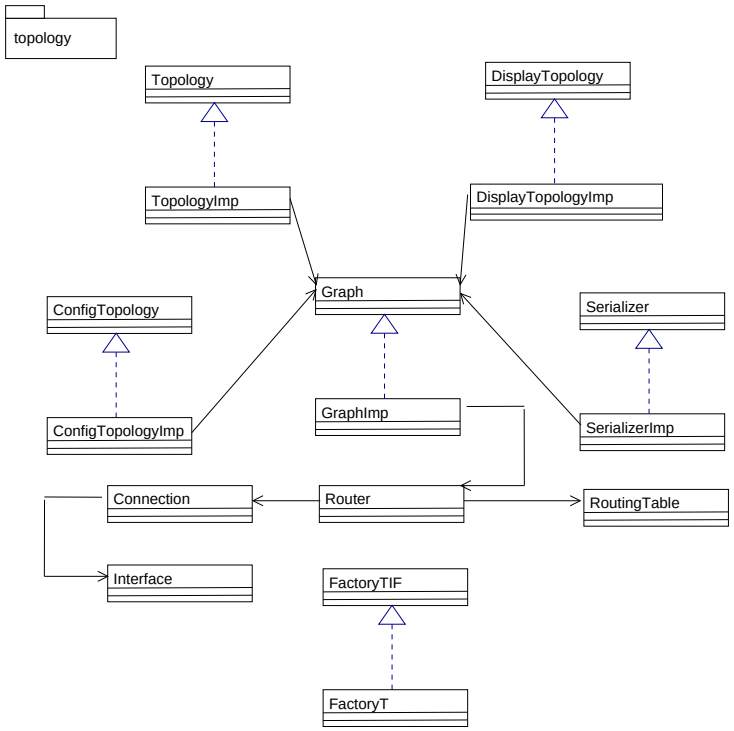

Figure 6.4: Topology classes overview

```
package topology;
import java.awt.*;
public interface Graph{

public Graph getGraph();
public Router getFirstRouter();

public void addRouter(Point p);
public void delRouter(int info);
public void updateRouter(Router router);
public Router getRouter(int number);

// finds edge from router1 to router2
public Connection getConnection(Router router1, Router router2);

//finds edge from info1 to info2
public Connection getConnection(int info1, int info2);

        // used for removing connections
        //a connection can only be removed, if the interface is not configured
        //or has been deleted
        public boolean hasIfConfigured(int router1,int router2);

//finds edge with info
public Connection getInfoConnection(int vertice,int info);

public void setConnection(int info1, int info2);
public void delConnection(int number1, int number2);
}
```

Router

Instances of `Router` contain the router configuration information. They also maintain a list of Connection objects, which represent the router interfaces.
The Router objects are created and accessed through the Graph object.
The class Router is responsible for numbering the routers and ensure that the numbers are unique. For this the router has a static variable maintaining the number of the last created router and a static Stack variable for storing the numbers of deleted routers. When a new router gets created, it gets a number from the stack if the stack is not empty, else if the stack is empty the number is incremented by one. When a router gets deleted, its number is put on the stack. Each Router object is responsible for the labels of its interfaces.The interface labels are managed the same way as the router labels. Every Router object has a variable holding the number of the last created interface and a stack to hold the labels of the deleted interfaces. Every Connection object of a router is identified by its interface number. Notice that the Connection object may also

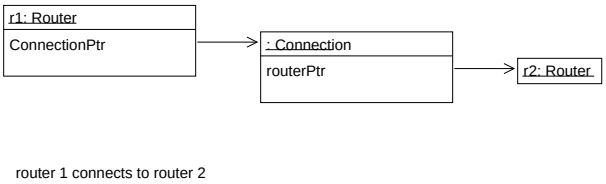

Figure 6.5: Connecting routers

have an `Interface` object, which is not related to the interface number
For the source code see A.2.1, page 59

Connection

As we have seen in the previous paragraph, the Router class maintains a list of Connection objects. A Connection object is added to the list when a user connects two routers in the CHANGE NETWORK PAGE. The new Connection object has a reference to the connected router. For an illustration see fig. 6.2.3. Notice that both routers get a Connection object, they are linked in both directions. The Connection objects are linked together, for this the Connection has a reference to the next Connection object. A Connection object represents an interface for a router and also maintains a link to the router which is reachable through this interface. This interface can be configured or unconfigured, if it is unconfigured, the field `iface` is empty (null), else it contains an initialized `Interface` object. Connection has a method `createIf()` which creates and initializes the Interface object.

For the source code see A.2.2, page 62

Interface

The Interface object represents a configured interface. When the user configures an interface of a router process and the according router from the topology-view is updated the Interface object gets created and initialized or modified. The class Interface has fields for the Ip number, the netmask, the fifo number and the fifo direction. For the source code see A.2.3, page 64

Topology, ConfigTopology, DisplayTopology, Serializer

There are several classes which use the Graph class, those are the classes Topology, ConfigTopology, DisplayTopology, Serializer. Those classes can be seen as providing different views on the graph class. This allows to modularize the access to the graph and allows to have different access methods with the same name and signature but different semantics. For example the method getConnections in DisplayTopology returns only the routers numbered (x + n) for router n

whereas the method getConnections in ConfigTopology returns all routers connected to router x. The drawback of not having just one class updating Graph is that a programmer has to be aware of the different references to a Graph instance.

Serializer

The class `SerializerImp` is used for the serialization of Java `Objects`. For an introduction to serialization see [Hor00]. We use serialization to save Objects to a binary file and we use it to send Objects over a TCP connection as a byte stream. A Graph object cannot be serialized by extending the Serializable interface, as it is possible for most other objects. The `graph` has to be transformed into a Java.util.Vector object. The method `Vector serialize(Graph graph)` [5]

takes a Graph object as argument and returns a Vector object. This Vector contains all the objects that were in Graph, like Router-, Connection-, Interface- and RoutingTable objects. The objects referenced by Graph all implement the Serializable interface and need no or only minor changes (class Router) to get serialized. The Vector object can be serialized by default. The method `Graph deSerialize(Vector v)`restores the Graph object from a serialized Vector object.

Factory

Class Factory initializes the `topology` interface classes with the classes that implement the interfaces. As an example see the use of Factory in the class Allocate from package allocate:

```
Topology t = factory.getTopology();
DisplayTopolgy displayT = factory.getDisplayTopology();
ConfigTopology configT = factory.getConfigTopology();
```

If one wants to use another class to implement the interface only the factory class has to be changed. This makes the program more independent from the representation of the data. Notice that it is not an implementation of factory pattern [Gra98] it is just used to modularise the program and facilitate the change of the topology implementation, whereas the factory pattern is used to extend a framework.
source code of class Factory:

```
package topology;
public class FactoryT implements FactoryTIF{
```

[5] the method signature is sometimes written in Java syntax `return-type method(Type:Object)`or we use the `UML` notation `method(Object:Type):return-type`

```
    public Graph getGraph(){
        return(new GraphImp());
    }
    public Topology getTopology(){
        return(new TopologyImp());
    }
    public DisplayTopology getDisplayTopology(){
        return (new DisplayTopologyImp());
    }
    public Serializer getSerializer(){
        return(new SerializerImp(getGraph(),getTopology()));
    }
    public ConfigTopology getConfigTopology(){
        return(new ConfigTopologyImp());
    }
}
```

6.2.4 Package Allocate

The responsibility of the allocate classes is to start router processes upon a command from the CREATE NETWORK PAGE. For a classes overview see figure 6.6. An instance of class Allocate is initialized when the class Administrator gets the command allocate. Administrator checks if there are router processes to start. For this it calls the method getRouterNames() of the Allocate object, which will return the names of the routers contained in topology. Administrator compares the names to the list of running processes that it maintains. If there is a process to start is calls the method getVrc (routerName: String) of class Allocate. This method will create an initialization file for the router, start the router and return a handle for it. For details concerning the administrator see section administrator 5.14.

Data

Data is the class which is responsible for retrieving the topology information associated to a specified user-ID. Its main method is `Graph get(userID:String)`, which uses a `TopologyFile` object to read the file.

When Allocate is initialized it retrieves the topology information from the user which has send the command to the administrator. Object Allocate uses methods from the class Data for this.

Command

Command is a class that gets initialized with topology classes. It is used to extract the configuration information for a certain router. It does this with the method `StringBuffer get(routerName:String)` which returns a StingBuffer

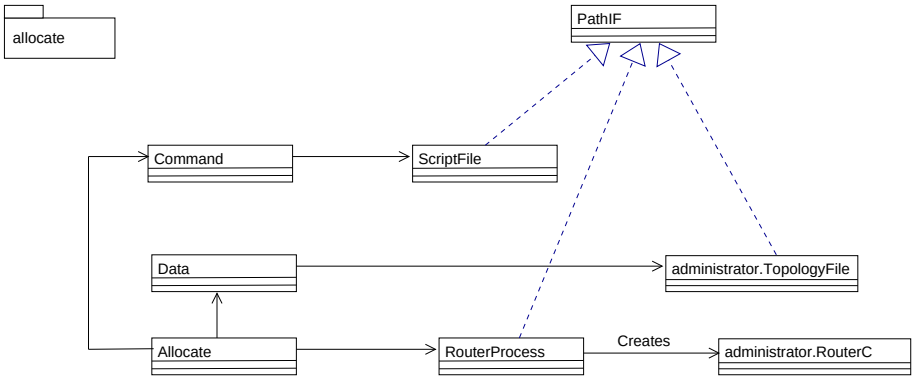

Figure 6.6: Allocate classes overview

object containing router commands. The Allocate instance which has called this method then writes the StringBuffer to a file, using methods of a ScriptFile object.

RouterProcess

The class RouterProcess starts router processes. It has a handle for the Java.-system.Runtime instance which enables it to execute shell commands. To start a router it takes a random number in the range of possible non-public port numbers and tries to start a router with this port number as argument. The second argument to this command is a script file from which the router configures itself. If the router is started for the first time (no information could be extracted from topology) the file is empty. Java allows to get an InputStream for the created Process. We read from this stream and if it is empty the creation of the process failed. When the start succeeded, RouterProcess creates an administrator.RouterC object. It is initialized with the port number and the routername. RouterC is mainly a TCP connection Socket for the newly created router process. The communication with the router process will take place over this connection.

The RouterC object is returned to Allocate from where it is returned to class Administrator.

PathIF

This interface is implemented by the classes TopologyFile, RouterProcess and ScriptFile. The interface is not used for some object oriented design purposes, it is just a way to centralize the configuration data that the administrator needs. This file has to be updated when the program administrator is installed.

6.2.5 Package Administrator

In this section we look at the class Administrator and its inner class AdminThread and at the package administrator. The class Administrator itself is not contained in the package. For an overview see figure 6.7. Class Administrator is the main class of the program administrator. The administrator program gets started by calling the main() method of class Administrator.

Class Administrator

Administrator handles the connection requests from the applets. It loops forever, listening for client connection requests on a ServerSocket. When a request comes in, Administrator accepts the connection, reads the action-command it gets from the applet and creates a new AdminTread object to process it, hands it a socket and starts the thread. Then Administrator goes back to listening for connection requests. An important member of Adminstrator is java.util.Map. All users having router processes are contained in this map. The map contains userID - RouterCollection pairs. A RouterCollection object is a collection of RouterC objects. Notice that we establish a TCP connection to the router processes just after they got started. The connection remains until the user deletes the router or the user terminates the session.

Class AdminThread

AdminThread is implemented as inner class [Fla99] of the Administrator class and has thus access to all the members of the Administrator class. AdminThread runs as a thread started from the Administrator class. AdminThread analyzes the action-command (described in section 5.3.4) and calls the appropriate function. For an overview of the administrator functions see figure 5.14 .

Administrator the Package

Administrator is also a package, package `administrator`. It contains the classes for networking and class Fifo, used by classes from the `config` package to create fifos. It also contains the classes RouterCollection and RouterC.

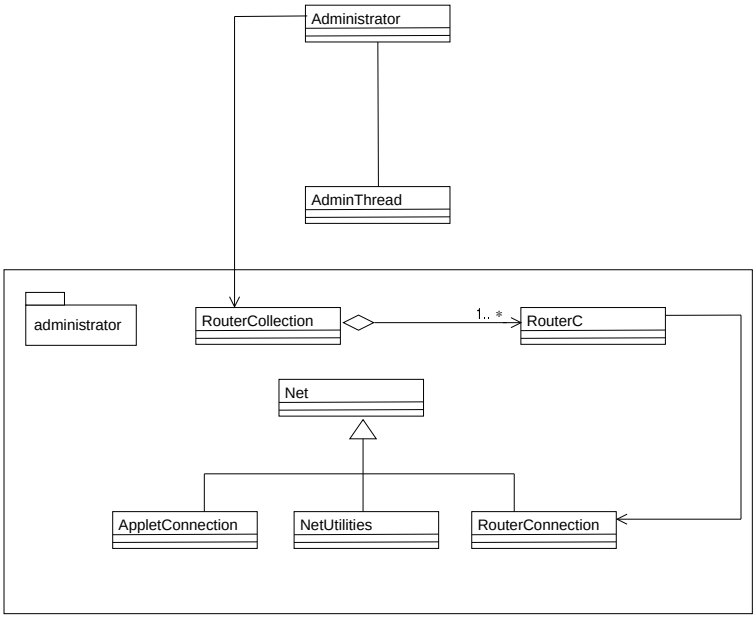

Figure 6.7: Administrator classes overview

Chapter 7

Results

7.1 Properties of the Application

7.1.1 Features

Microvar routers have first to be configured before they can be connected. This property of the microvar routers is transparent to the user, a user has the feeling to first connect the routers and then to configure them. The configuration state of the routers and the topology is saved in each session, allowing the user to continue the exercise where (s)he has interrupted it. The configuration commands are simple commands, microvar specific configuration commands are hidden from the user. A user has to log in with its name and password, allowing only authorized users to use the program. Several users can log in concurrently and do the exercise at the same time.

Several messages for the microvar configuration can be bundled to one message. To add such a bundled command set only a class has to be added to the application. This is very useful if a user wants to configure many routers manually.

7.1.2 Drawbacks

The allocation of routers on different hosts for the same user is not yet possible. It would require from the user to choose for every router on which host (s)he wants to run the VR. But on the other hand the topology from the underlying hosts should be transparent to the user. A solution would be, that the system allocates automatically the routers on different hosts. This would only be necessary for big topologies. No measurements have been made in this work about the upper limit of the amount of VR's on one host, but so far on machines with few resources twenty VR's are running concurrently. If not for large topologies we don't need two or more hosts for one user.

Allocating routers on a different host as the host where the administrator is installed is not implemented yet. An approach would be to send the configura-

tion scripts for the routers to the host, so that the routers could read them at startup and then the routers could be started remotely.

An easier approach would be to have the administrator program and the routers on the same host but on a different host as the web server. Then we only would have to redirect the TCP connections from the web server host to the host where the administrator and the routers are. The implementation of this feature would increase the scalability of the application.

7.1.3 Extensibility

For every configuration command which can be transmitted to a microvar router at run time, the application can easily be extended to support it. The application could also be extended, with no big overhead, to support commands directly sent to the API. Also adding an applet to the application, or adding a button with some new functionality to an applet, can be done easily.

The concept of this application could also be used to establish a web-based communication to other processes than virtual routers. One could think of any processes being able to establish a TCP connection and performing some action upon receiving a command.

Some concrete examples of functionality which could be added to the application are the following:

Add a function to write a router configuration file to a location where it could be processed by a script.

Add an applet that displays graphically the configured IP numbers and routing tables of the routers.

Integrating Wivrec with the IP Network Simulation Module

The application developed in this thesis can be extended to be used for the IP Network Simulation Module or it can serve as prototype for this module.

If it would be used as module security issues would have to be considered.

The application would also have to be adapted to the remote network laboratory architecture described in [Bra]. Then the application would have to be enhanced to allow the use of predefined configuration scripts. It would also have to be updated to allow to analyze the student working results efficiently. Both tasks could be done with simple tools like shell scripts.

The underlying virtual router system would also have to be thoroughly tested. This application and maybe also the underlying virtual routers system would have to be updated so that the router commands match exactly the semantics of commands to real routers.

Chapter 8

Summary and Outlook

The appliction described in this thesis could be enhanced to be used as an IP Network Simulation Module. The IP Network Simulation Module is a learning module developed for the Virtual Internet and Telecommunications Laboratory. On the other hand the application is a remote interface to virtual routers in its own right and could be used as such to remotely configure routers. The set of supported router commands could of course be enlarged. This would for example enable a user to configure a test bed for network simulation remotely.

At last the application could also be used to start other processes than router processes and thus be used as a remote tool to configure and control such processes. The set of commands and the topology would of course have to be adapted. An application area for such processes would be simulations where processes running independently of each other are needed.

Appendix A

Appendix

A.1 Example: A Topology with 3 Routers

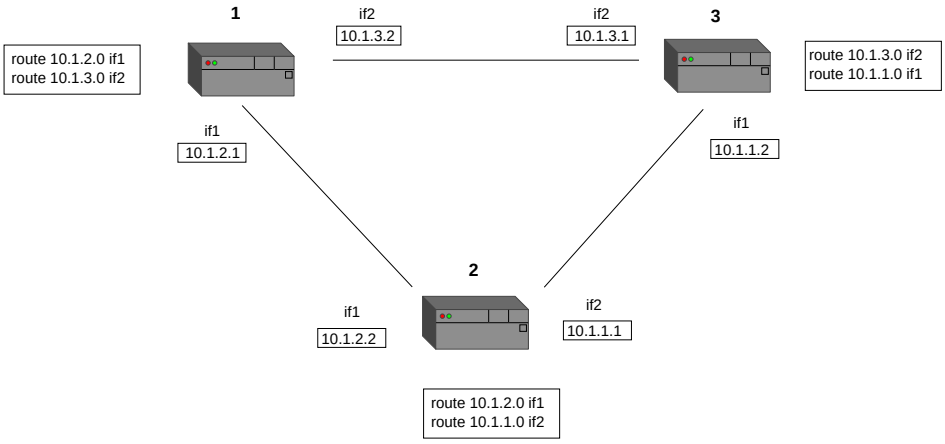

Figure A.1: Example overview

This is a little example for a network topology with 3 VR's for one user. We have configured the network topology which was depicted on figure 4.2 in chapter User View. First we configured router 1, then router 2, and at last router 3. The user commands were entered in the listed order. The session was then terminated. On a next session the routers were started again with the configuration scripts described in subsection A.1.2

See the following subsection for the user commands to the routers.

A.1.1 User Commands

User Commands for Router 1

Router 1
Configuring the Interfaces
`ifconfig add if1 10.1.2.1 255.255.255.0` `ifconfig add if2 10.1.3.2 255.255.255.0`
Configuring the Routing Table
`route add 10.1.2.0/24 if1` `route add 10.1.3.0/24 if2`

User Commands for Router 2

Router 2
Configuring the Interfaces
`ifconfig add if1 10.1.2.2 255.255.255.0` `ifconfig add if2 10.1.1.1 255.255.255.0`
Configuring the Routing Table
`route add 10.1.2.0/24 if1` `route add 10.1.1.0/24 if2`

User Commands for Router 3

Router 3
Configuring the Interfaces
`ifconfig add if1 10.1.1.2 255.255.255.0` `ifconfig add if2 10.1.3.1 255.255.255.0`
Configuring the Routing Table
`route add 10.1.1.0/24 if1` `route add 10.1.3.0/24 if2`

A.1.2 Configration Scripts

Configuration Script for Router 1

```
ifconfig add if2 10.1.3.2 255.255.255.0
ifconfig if2 qs create droptail
ifconfig if2 qs chain 0 1
ifconfig if2 qs chain 1 0
ifconfig if2 connect fifo 2 l
ifconfig add if1 10.1.2.1 255.255.255.0
ifconfig if1 qs create droptail
ifconfig if1 qs chain 0 1
ifconfig if1 qs chain 1 0
ifconfig if1 connect fifo 1 l
route add 10.1.2.0/24 if1
route add 10.1.3.0/24 if2
```

Configuration Script for Router 2

```
ifconfig add if2 10.1.1.1 255.255.255.0
ifconfig if2 qs create droptail
ifconfig if2 qs chain 0 1
ifconfig if2 qs chain 1 0
ifconfig if2 connect fifo 3 l
ifconfig add if1 10.1.2.2 255.255.255.0
ifconfig if1 qs create droptail
ifconfig if1 qs chain 0 1
ifconfig if1 qs chain 1 0
ifconfig if1 connect fifo 1 r
route add 10.1.2.0/24 if1
route add 10.1.1.0/24 if2
```

Configuration Scrpt for Router 3

```
ifconfig add if2 10.1.3.1 255.255.255.0
ifconfig if2 qs create droptail
ifconfig if2 qs chain 0 1
ifconfig if2 qs chain 1 0
ifconfig if2 connect fifo 2 r
ifconfig add if1 10.1.1.2 255.255.255.0
ifconfig if1 qs create droptail
ifconfig if1 qs chain 0 1
ifconfig if1 qs chain 1 0
ifconfig if1 connect fifo 3 r
route add 10.1.1.0/24 if1
route add 10.1.3.0/24 if2
```

A.2 code examples from package topology

A.2.1 topology.Router.java

```
;
package topology;
import java.io.*;
import java.util.*;
import java.awt.Point;

 public class Router implements Serializable{

    public Point p = new Point(0,0);
    public int number;
    public String name;
    public int ethernetCards;
    public RoutingT routingTable = new RoutingT();
    public transient Router nextRouter;
    public transient Connection  connectionPtr;
    public static Stack routerStack = new Stack();
    public static int routers = 0;
    // used to keep the static number while serializing
    private int routersSerialize = 0;
    private int[] routersArray = null;
    Stack cardStack = new Stack();

    public Router(Point p, Router nextRouter,Connection connectionPtr){

        this.nextRouter = nextRouter;
        this.connectionPtr = connectionPtr;
        this.p = p;
        number = addRouter();
        name = Integer.toString(number);
        ethernetCards = 0;
    }
    public int getNumber(){
        return(number);
    }
    public String getName(){
        return(name);
    }
    public Point getPosition(){
        return(p);
    }
    public void setPosition(Point p){
        this.p = p;
```

```
}
public Connection getConnection(){
    return connectionPtr;
}
public void addRoute(String ip,String nm,String ifs){
    routingTable.addRoute(ip,nm,ifs);
}
public void delRoute(String ip,String nm,String ifs){
    routingTable.delRoute(ip,nm,ifs);
}
public Vector getRoutes(){
    return(routingTable.getRoutes());
}
private void writeObject(ObjectOutputStream objectOut) throws IOException {
    serializeStack();
    routersSerialize = routers;
    objectOut.defaultWriteObject();
}
private void readObject(ObjectInputStream objectIn)
    throws IOException,ClassNotFoundException {
    objectIn.defaultReadObject();
    routers = routersSerialize;
    deSerializeStack();
}
public void serializeStack(){
    if(routerStack.size() > 0){
        routersArray = new int[routerStack.size()];
        for (int i = 0; i < routerStack.size();i++){
             routersArray[i] = ((Integer)routerStack.pop()).intValue();
        }
    }
}
public void deSerializeStack(){
    if(routersArray !=null){
        for(int i = (routersArray.length - 1);i>= 0; i--){
            routerStack.push(new Integer(routersArray[i]));
        }
    }
}
private  int addRouter(){
    int actualRouter = 0;
    if(routerStack.empty()){
        routers++;
        actualRouter = routers;
    }
    else{actualRouter = ((Integer)routerStack.pop()).intValue();
```

```
        }
        return actualRouter;
    }
    public void removeRouter(){
        if (this.number <= routers){
      routerStack.push(new Integer(this.number));
        }
        else{ e.error(" try to remove a nonexisting router");
        }
    }
    public int getEthernetCard(){
        return addEthernetCard();
    }
    private int addEthernetCard(){
        int actualCard = 0;
        if(cardStack.empty()){
            ethernetCards = ethernetCards + 1;
            actualCard = ethernetCards;
        }
        else{actualCard = ((Integer)cardStack.pop()).intValue();
        }
        return(actualCard);
    }
    public void removeEthernetCard(int card){
        // the card to remove exists or is already on the cardStack
        if (card <= ethernetCards ){
             cardStack.push(new Integer(card));
        }else{
     e.error(" try to remove a nonexisting EthernentCard");
        }
    }
}
```

A.2.2 topology.Connection.java

```
package topology;

public class Connection implements Serializable{

    public Interface iface;
    private int ethernetCard;
    public transient Connection nextConnection;
    public transient Router routerPtr;
    private int routerName;

    public Connection(int card,Connection nextConnection,
                              Router routerPtr,int name){
        this.nextConnection = nextConnection;
        this.routerPtr = routerPtr;
        ethernetCard = card;
        routerName = name;
    }
    public void setEthernetCard(int e){
        ethernetCard = e;
    }
    public int getEthernetCard(){
        return ethernetCard;
    }
    public int getRouter(){
        return routerName;
    }

    // interface commands
    public void createIf(int ifx, String ip){
        iface = new Interface(ifx,ip);
    }
    public void createIf(int ifx, String ip,String nm){
        iface = new Interface(ifx,ip,nm);
    }
    public int getIf(){
        if(iface != null){
            return(iface.ifx);
        }else {
    return 0;
        }
    }
    public String getIp(){
        return iface.ip;
```

```
    }
    public void setIp(String ip){
        iface.ip = ip;
    }
    public String getNm(){
        return iface.nm;
    }
    public void setNm(String nm){
        iface.nm = nm;
    }
    public void setFifo(int[] fifo){
        //iface.fifo = fifo;
        iface.setFifo(fifo);
    }
    public int[] getFifo(){
        if (iface != null){
            return iface.getFifo();
        }
        else{
            int[]fifo ={0,0};
            return fifo;
        }
    }
    public void deleteIf(){
        iface = null;
    }
}
```

A.2.3 topology.Interface.java

```
package topology;
import java.io.*;

public class Interface implements Serializable{

    int ifx;
    String ip;
    String nm = null;
    int[] fifo; // int[0]=fifonr //int[1]={0=left;1=right}
    String queue = null;

    public Interface(int ifx,String ip){
this.ifx = ifx;
this.ip = ip;
    }
    public Interface(int ifx,String ip,String nm){
this(ifx,ip);
this.nm = nm;
    }
    public void setFifo(int[]fifo){

this.fifo = (int[])fifo.clone();
    }
    public int[]getFifo(){
int[]f = (int[])fifo.clone();
return(f);
    }
}
```

A.3 code examples from package config

A.3.1 config.IfAdd.java

```
package config;
import administrator.*;
import topology.*;
import java.util.*;

public class IfAdd extends ConfigCommand {

    String error = "error: ";
    String help = " ifconfig add ifx <ip> [<nm>] \n";
    String result = null;
    Fifos fifos = Fifos.getInstance();

    public String exec(ConfigTopology configT,String
          routerName,RouterC r, String command) {

//since we are here, the "ifconfig add" syntax is so far correct
        //extract interface number of the  command
StringTokenizer getIf = new StringTokenizer(command);
//remove the two first tokens: "ifconfig" and "add"
String token = getIf.nextToken();
token = getIf.nextToken();
token = null;
int ifx = 0;
String ifs = null;
try{
    StringBuffer ifxs = new StringBuffer(getIf.nextToken());
    ifs = ifxs.toString();
    ifx = Integer.parseInt(ifxs.substring(2));
}
catch(Exception e1){
    //send message to user
    return(help);
}

//user isn't allowed to add this if
if(!configT.hasIf(ifx)){
    //send message to user
    return(error + "try to add unconnected if");
}
```

```
//user is allowed to add this interface
        //
//add interface to the routerprocess
//
r.send(command);
// possible results:
// <">"><Syntax error>
// <">"><number><ifx><ip><nm><0><connected>
// <">">
// <number>...
// <Syntax>...
String res = r.get();
if(res == null){
    return("no result string from router");
}
java.util.StringTokenizer st = new java.util.StringTokenizer(res);
String token1 = null;
token1 = st.nextToken();//prompt ,syntax or number,
if(token1.equals(">")){
    if(st.hasMoreTokens()){
token1 = st.nextToken();//token syntax error or number
    }
    else{
return("no result string from router");
    }
}
if((token1.equalsIgnoreCase("Syntax"))){
    return(help);
}

//remove token with if number (we have it already)
token1 = st.nextToken();token1 = null;
String ips = st.nextToken();e.debug("ips: " + ips);
String nms = st.nextToken();e.debug("nms: " + nms);

//if the user didn't provide a
//netmask, a default netmask is added by the router

//update the topology with the netmask
configT.createIf(ifx,ips,nms);
result = " if" +ifx + " " + ips + " " + nms + " created";
//connect the interface to the router
r.send("ifconfig " + ifs + " qs create droptail");
String buffer = null;
buffer = r.get();
r.send("ifconfig " + ifs + " qs chain 0 1");
```

```
buffer = r.get();
r.send("ifconfig " + ifs + " qs chain 1 0");
buffer = r.get();

//connect the interface to interface of other router
int[]fifo = configT.getOtherFifo(ifx);
if(fifo[0]== 0){
    int newFifo =fifos.getFifo();
    fifo[0] = newFifo;
    fifo[1] = 0;           //int[1]={0=left;1=right}
    //create always left fifo first
    r.send("ifconfig " + ifs + " connect fifo " + fifo[0] + " l");
}
else{
    fifo[1]=1;
    r.send("ifconfig " + ifs + " connect fifo " + fifo[0] + " r");
}

buffer = r.get();
buffer = null;
//update this if with created fifo
configT.setFifo( ifx,new int[]{fifo[0],fifo[1]});
return(result);
    }
}
```

A.4 Administrator.java

```
import config.*;
import administrator.*;
import allocate.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.StringTokenizer;

public class Administrator {

    Map m = new HashMap();
    Parameter p;
    public  int appletPort = portNumber;
    ServerSocket serverSocket = null;
    String userID = null;
    String use = null;

    public void start(){
try{
    // wait for connection from applet
    serverSocket = new ServerSocket(appletPort);
    //  flags = flags | CREATE;
    // accept the connection from the applet and create a
                                        socket for the connection
    while(true){
try {
    //("waiting for connection");
     Socket socket = serverSocket.accept();
    //("connection accepted");
    administrator.AppletConnection a;
    a = new administrator.AppletConnection(socket);
    //"new appletConnection created");
    a.open();
    p = new Parameter(a.get());
    userID = p.getParameter("userID");
    use = p.getParameter("use");
    //AdminThread handler;
    new AdminThread(socket,a,userID,use).start();
 }catch (IOException e1) {
      e.error(e1.getMessage());
      return;
}
    }
```

```
} catch (Exception e1) {
    e1.printStackTrace();
    e.error("an e.error in the client occured." );
    return;
}
    }
    public static void main(String[] args) throws IOException {
        Administrator a = new Administrator();
        a.start();
    }

    class AdminThread extends Thread{

        Socket s;
        String userID;
        String use;
        administrator.AppletConnection a;
        public AdminThread(Socket s,administrator.AppletConnection a,
                           String userID,String use){
            this.s = s;
            this.a = a;
            this.userID = userID;
            this.use = use;
        }
        public void run(){
    try{
    if (use.equals("save")){
            save(userID);
        }
    else if (use.equals("retrieve")){
         retrieve(userID);
    }
    else if(use.equals("create")){
         allocate(userID);
    }
    else if(use.equals("close")){
         close(userID);
    }
    else if(use.equals("exit")){
         exit(userID);
    }
    //
    // cases for configuration
    //
    else if (use.equals("configure")){
                  configure(userID);
```

```
        }
    } catch (Exception e1) {
        e.error("thread errror" + e1.getMessage());
        return ;
    }
     }

//closes the TCP connection of the routers while keeping them running
//used for testing
private void exit(String userID){

     RouterCollection rC = (RouterCollection)m.get(userID);
     rC.exitAll();
}
//shut down routers
private void close(String userID){

    RouterCollection rC = (RouterCollection)m.get(userID);
    rC.delAll();
    a.send("ack");
    a.close();
}
private void save(String userID)

    a.send("ack");
    TopologyFile fileT = new TopologyFile(userID);
    fileT.openWrite();
    fileT.saveObject(a.getObject());
    a.send("ack");
    a.close();
}
private void retrieve(String userID){

    String ack;
    TopologyFile fileT = new TopologyFile(userID);
    if(fileT.openRead()){
        a.send("0");
        ack = a.get();
        a.sendObject(fileT.getObject());
        a.close();
    }
    else{
a.send("-1");
ack = a.get();
a.close();
    }
```

```
    }

    private void allocate(String userID){

        FactoryTIF factory = new FactoryT();
        Allocate alloc = new Allocate(userID,factory);
        RouterCollection rC;
        rC = (RouterCollection)m.get(userID);
        if(rC == null){
             rC = new RouterCollection();
        }
        Vector topo = alloc.getRouterNames();
        String item;
        //
        //processes that don't match a router in the topology are shut down
        Enumeration running = rC.getProcessNames();
        while(running.hasMoreElements()){
    item = (String)running.nextElement();
    if(!topo.contains(item)){
                rC.del(item);
    }
        }
        //processes get started for routers in the topology which don't have
        //a matching running process
        Enumeration topoEnum = topo.elements();
        item = null;
        while(topoEnum.hasMoreElements()){
            item = (String)topoEnum.nextElement();
            if(!rC.contains(item)){
                rC.add(item,alloc.getVrc(item));
            }
        }
        m.put(userID,rC);
        a.send("routers allocated");
        a.close();
    }
    private void configure(String userID){
         String routerName = p.getParameter("routerName");
a.send("ack");
FactoryTIF factory = new FactoryT();
RouterCollection rC = (RouterCollection)m.get(userID);
RouterC r = rC.get(routerName);
//give Appletconnection
Dialog d = new Dialog(userID,r,factory,a);
d.start();
a.close(); }}}
```

Bibliography

[Apa] Apache http server. http://httpd.apache.org.

[BBa] Florian Baumgartner and Torsten Braun. Quality of service and active networking on virtual router topologies. *The Second International Working Conference on Active Networks*.

[BBb] Florian Baumgartner and Torsten Braun. Virtual routers: A novel approach for qos performance evaluation. *Quality of future Internet services, QofIS'2000*.

[Bra] Marc-Alain Steinemann Stefan Zimmerli Thomas Jampen Torsten Braun. Architectural issues of a remote network laboratory. *have to look it up...*

[FB] Torsten Braun Florian Baumgartner. Virtual router manual and api description version 1.9. *Techical Report,IAM-01-001*.

[Fla99] David Flanagan. *Java in a Nutshell,Third Edition*. O'Reilly and Associates, 1999.

[Gra98] Mark Grand. *Patterns in Java: a catalog of reusable design patterns illustrated with UML*. John Wiley and Sons,Inc, 1998.

[Hor00] Ivor Horton. *Beginning Java 2 - JDK 1.3 Edition*. wrox, 2000.

[HTM] Html 4.0 specification. http://www.w3.org/TR/REC-html40.

[Java] Java api specification. http://java.sun.com.

[Javb] Sun java tutorial. http://java.sun.com/docs/books/tutorial.

[Lin] Debian linux. http://www.debian.org.

[PHP] Php hypertext preprocessor. http://www.php.net.

[SVC] Swiss virtual campus. http://www.virtualcampus.ch/.

[VIT] Virtual internet and telecommunications laboratory of switzerland. http://www.vitels.ch.